SOFTWARE DEVELOPMENT FOR MAN-MACHINE INTERFACE FOR AN INDUSTRIAL ROBOT

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES
OF
THE MIDDLE EAST TECHNICAL UNIVERSITY

BY

MAHİR CİHAN CENGİZ

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF
MASTER OF SCIENCE

IN

THE DEPARTMENT OF MECHANICAL ENGINEERING

DECEMBER 2003

Approval of the Graduate School of Nature 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. Kemal İder
Head of Department

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

Examining Committee Members:

Prof. Dr. Bilgin Kaftanoğlu
Supervisor

Prof. Dr. Kemal Özgören	______________________
Prof. Dr. Bilgin Kaftanoğlu	______________________
Prof. Dr. Tuna Balkan	______________________
Assist. Prof Dr. İlhan Konukseven	______________________
Prof. Dr. İsmet Erkmen	______________________

ABSTRACT

SOFTWARE DEVELOPMENT FOR MAN-MACHINE INTERFACE
FOR AN INDUSTRIAL ROBOT

CENGİZ, Mahir Cihan
M.S., Department of Mechanical Engineering
Supervisor: Prof. Dr. Bilgin KAFTANOĞLU

December 2003, 104 Pages

In this study, a robotic software, which controls the robot, is developed. The robot considered is a six degree of freedom robot and it is designed and manufactured in METU. User can send the robot anywhere in space within its workspace, in any orientation. Forward and inverse kinamatics can be executed according to the needs.

Simulation framework is embedded into the software for the 3D visualisation of the robot. Any movements can be simulated on the screen.

Software also generates the path for the given points. Then generated path is simulated on the screen. All position, velocity and acceleration graphics of joints can be examined for the generated path.

Keywords: Industrial robot, kinematic analyses, 3D simulation, path generation.

ÖZ

ENDÜSTRİYEL BİR ROBOT İÇİN İNSAN-MAKİNA
ARAYÜZ PROGRAMININ GELİŞTİRİLMESİ

CENGİZ, Mahir Cihan
Yüksek Lisans, Makina Mühendisliği Bölümü
Tez Yöneticisi: Prof. Dr. Bilgin KAFTANOĞLU

Aralık 2003, 104 Sayfa

Bu tez çalışmasında, endüstriyel bir robotu kontrol eden bir yazılım geliştirilmiştir. Bahsi geçen robot altı serbestlik dereceli olup ODTÜ'de tasarlanmış ve imal edilmiştir. Kullanıcı, bu programla, robot kolunu çalışma hacmi içinde istediği pozisyon ve yönelime gönderebilir. Pozisyon kontrolleri için ileri ve geri kinematik analizler yapılmıştır.

Robotun üç boyutlu görünümünü sağlamak için programa simulasyon iskeleti gömülmüştür. Bu sayede robotun her hareketi ekranda simüle edilebilmektedir.

Program ayrıca verilen noktalar için rota oluşturabilmektedir. Bu rota ekranda izlenebilmektedir. Ayrıca, rota içinde eksenlerin pozisyon, hız ve ivme grafikleri de izlenebilmektedir.

Anahtar Kelimeler: Endüstriyel robot, kinamatik analiz, 3 boyutlu simulasyon, yörünge oluşturma.

TABLE OF CONTENTS

List of Tables

List of Figures

Nomenclature

a_{i+1}	Length of link $i+1$, distance between u_3^{i} and u_3^{i+1} along u_1^{i}.
a_2, a_3	Constant link lengths.
α_{i+1}	Twist of link $i+1$, angle from u_3^{i} to u_3^{i+1} about u_1^{i+1}.
C	Rotation matrix of the orientation of end-effector.
C_{i-1}^{i}, $C_{(i-1)(i)}$	Rotation matrix of link i with respect to link (i-1).
c_{ij}	elements of the C matrix. (i=1,2,3 & j=1,2,3).
C_f	Coefficient matrix of the trajectories.
d_{i+1}	Offset of link $i+1$, distance from the origin of frame i to u_1^{i+1} along u_3^{i}.
d_4, d_5	Constant link offsets.
d_P	Length of the end-effector.
$F_n(x)$	The functions of the trajectories planned in the via points.
J_P	Jacobian matrix.
J_{PX}, J_{AX}	Submatrices of the J_P.
M	Transformation matrix in the trajectory planning.
P	Position vector of the tip point w.r.t. the base.
P_1, P_2, P_3	elements of the P vector.
Q	Angular position vector.
R	Position vector of the wrist point w.r.t. the O_0.
R_1, R_2, R_3	elements of the R vector.
R_x, R_y, R_z	Rotation matrices about the principal axes.

θ_{i+1}	Angle of joint *i+1*, angle between u_1^i and u_1^{i+1} about u_3^i.
θ, φ, ϕ	Euler angles to determine the rotation matrices.
V	Matrix of the positions, velocities and accelerations of the points in the trajectory planning
V_P	Linear velocity vector of the tip point.
ω_P	Angular velocity vector of the tip point.
X_n, u_1^n	Unit vector of frame *n* in the direction of x-axis.
Y_n, u_2^n	Unit vector of frame *n* in the direction of y-axis.
Z_n, u_3^n	Unit vector of frame *n* in the direction of y-axis.

CHAPTER 1

INTRODUCTION

1.1 Introduction

In today's world, the need for speed and accuracy in production has become more and more important. Especially, in industry, the productivity and having good quality is very important. Therefore, computer controlled machines have been used for years. More and more of the loading/unloading tasks have been executed by the robots in recent years. In other words, industrial robots are beginning to revolutionise the industry. Robots are now useful in a wide variety of industrial applications, such as material handling, painting, welding, etc. In most of these applications, the operation of the robots are cheaper, faster and less dangerous.

1.2 What is Robot?

The term robot comes from Czech and means "forced labour". The term in its present interpretation was invented by the Czech writer Karel Capek in his satirical play R.U.R. "Rossum's Universal Robots". He depicted robots as machines, which resembled people but worked tirelessly [7]. Robot is defined in dictionaries as "an automatic device that performs functions ordinarily ascribed to human beings". This definition is true but not sufficient.

A robot can have both an automation and intelligence. It has automation that it is a machine that can control, in some degree or other, its own actions. It is a general manipulator in the sense that it is a machine built with the capacity to do many different things, perform many different intelligent actions. Robotics is a field of interest that combines theory and application, ideas with actual practical machine [5].

More explanatory definition for the industrial robot is given by the Robot Institute of America, “A robot is a reprogrammable multi-functional manipulator designed to move material, parts, tools or specialised devices, through variable programmed motions for the performance of a variety of tasks.” An industrial robot is a general-purpose manipulator consisting of several rigid links connected in series by revolute or prismatic joints. One end of the chain is attached to a supporting base, while the other end is free and attached with a tool to manipulate objects or perform assembly tasks. The motion of the joints result in the relative motion of the links.

The motion of the end-effector is generated by controlling the position and velocity of the robot’s axes of motion. Basically the robot needs six axes of motion (or degrees of freedom) to reach an arbitrary point with a specific orientation in space. A different orientation might completely change the position of the robot arm. For example, to place a weld on the top side of the beam below requires completely different orientation from that required to place a weld at almost the same point but on the bottom side of the beam (Figure 1.1).

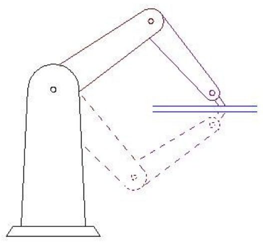


Figure 1.1 Welding Robot

1.3 History of Robots

1.3.1 Robots created by humans

Until a mechanical device with reprogrammable and multifunction capabilities emerged as an idea (in the 1920s), there was no conscious history of robotics. But, we can track the idea of robots, as we are using the term, in myths and continuing up to recent times.

In this part, we will consider real human beings that created mythical beings in the sense that there is no evidence that these “real” people did in fact create workable robots. Empedocles, a philosopher living in the 5th century BC, is the first human being to be credited with having made an isomorph, an animated statue. The next example comes from sometime in 12th century AD. Albartus Magnus, a priest, was said to have spent over 20 years constructing a robot made of wood, metal, wax and leather that was fully mobile and could welcome visitors at his door and speak to them. According to the legend, the fellow who is now Saint Thomas Aquinas is said to have destroyed Magnus’ robot on the grounds it was the work of devil. Table below summarises the chronology of these and other alleged creations. (Table 1.1)

Table 1.1 Chronology of Alleged Creations

CHRONOLOGY	REAL HUMANS	MYTHICAL CREATIONS
5th C. BC	Empedocles	Animated statue
12th C.	Albartus Magnus	Servant girl
13th C.	Bacon	Talking head

16th C.	Loew (used a formula to bring a clay robot to life)	Golem (animated clay robot)
16th C.	Paracelsus	Little man
17th C.	Goethe	Robot
19th C.	Anderson	Mechanical birds

1.3.2 Mechanical robots up to 1922

Around 1500 BC, Egyptian water clocks supposedly used human figurines to strike hour bells. The third century BC in Hellenic Egypt was a time of the development of many automated machines. All over the place there were statues, which were said to be able to speak, gesture and prophesy. In the second century BC, Philo is said to have made an even more elaborate theater that could go through five whole acts of a play from beginning to end.

In the first century, Petronius Arbiter created a doll that could move like human being. In 1557, Giovanni Torriani made a wooden robot for an Emperor that could fetch his daily bread from the store.

Vaucanson (1709-1782) created in 1738 a mechanical duck that could eat, excrete passable iso-olfactoric excrement, walk, quack, and do various things except fly. Later on, Vaucanson constructed a flute player that could play many different pieces of music. Another example like that is the writer created by Droz (1721-1790) that could write a one-page letter and then signs its name at the end.

The most elaborate mechanism of the 19th century is created in 1875. It is J.N. Maskelyne’s Psycho. This was a mechanical “half-man” sitting on the desk, which could nod his head and perform mathematical operations. Psycho could also do some low-level conjuring tricks, and could play whist[1]. The table below lists the real robots created from the birth of Christ. (Table 1.2)

Table 1.2 Chronology of Real Robots

TIME	CREATOR	ROBOT
1st C.	Petronius Arbiter	Moving Doll
16th C.	Leonardo	Mechanical man
16th C.	Giovanni Torriani	Walking robot
18th C.	Vaucanson	Flute player, Mechanical duck
18th C.	Droz	Writer robot
19th C.	Edison	Talking Doll
19th C.	Maskelyne	Psycho

1.4 Types of Industrial Robots

Industrial robots are widely used in manufacturing and assembly tasks such as simple material handling, spot/arc welding, parts assembly, and spray painting. They are used in space and undersea applications, and in hazardous applications. The manipulator is composed of the main frame (the arm) and the wrist, each having three degrees of freedom, or axes of motion. Structurally, the robots can be classified according to the coordinate system of the main frame:

- **Cartesian:** Three linear axes.

The main frame of cartesian coordinate robots consists of three orthogonal linear axes. An important feature of cartesian robots is equal and constant spatial

[1] A card game, similar to bridge, that involves probabilities and strategic skills

resolution, that is, the resolution is fixed in all axes of motion and throughout the work volume of the robot arm, but the robot lacks mechanical flexibility; it cannot reach objects on the floor or reach points invisible from its base. (Figure 1.2)

- **Cylindrical:** Two linear and one rotary axis.

The main frame of cylindrical coordinate robots consists of a horizontal arm mounted on a vertical column which, in turn, is mounted on a rotary base. The resolution of the cylindrical robot is not constant and depends on the distance between the column and the gripper along the horizontal arm. (Figure 1.3)

- **Spherical:** One linear and two rotary axes.

The kinematic configuration of spherical, or polar, coordinate robot arm is similar to the turret of a tank. It consists of a rotary base, an elevated pivot, and a telescoping arm, which moves in and out. The disadvantage of spherical robots compared with their cartesian counterparts, is that there are two axes with relatively low resolution that varies with the arm length. (Figure 1.4)

- **Articulated or Jointed:** Three rotary axes.

Articulated robots consist of three rigid members connected by two revolute joints and mounted on a rotary base. This kinematic arrangement closely resembles that of a human arm. Since the articulated robot has three rotary axes, its spatial resolution depends entirely on the arm position. The accuracy of an articulated robot is poor since the joint errors are accumulated at the end of the arm. On the other hand, it can move at high speeds and has excellent mechanical flexibility, which make it the most common small- and medium-sized robot. (Figure 1.5)

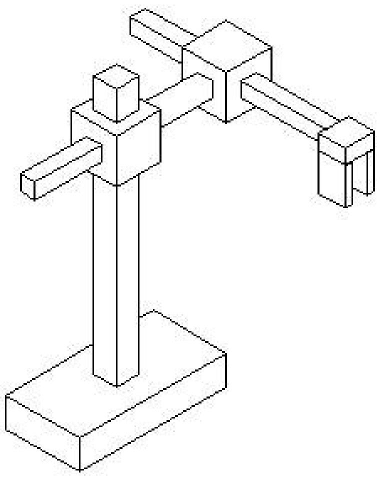


Figure 1.2 Cartesian Robot

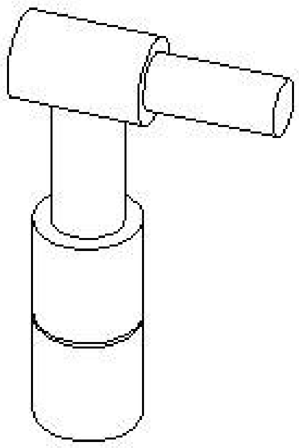


Figure 1.3 Cylindrical Robot

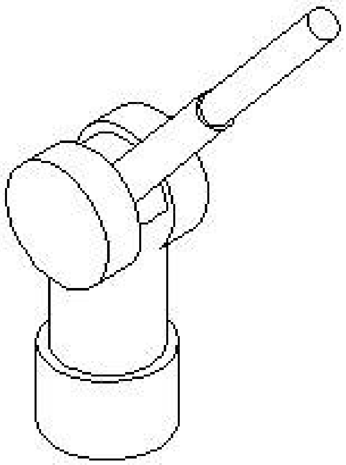


Figure 1.4 Spherical Robot

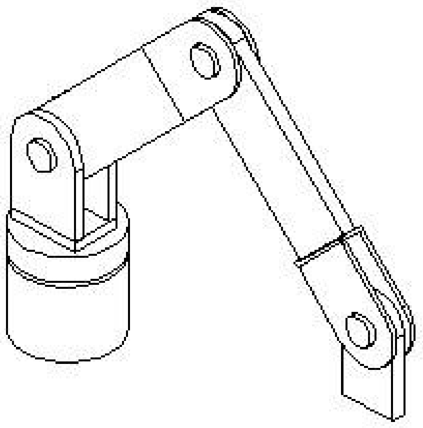


Figure 1.5 Articulated Robot

1.5 Driving Motors of Robot

The manipulator joints can be driven directly or indirectly. With direct drive, the joint shaft is coupled to the rotor of the drive motor. With indirect drive, the joint is connected to the drive motor through a transmission mechanism. Direct drive might provide better positioning accuracy since the intermediate gearing is eliminated and consequently the mechanism is free of backlash and hysteresis. But the main drawback of direct drive manipulators is that the motors, which drive the joints, are themselves a load for the motors at the lower joints (i.e. joints closer to the base). The leadscrew mechanism is used in most of the robots in recent years. Comparing this to other gearing systems, such as worm gear or harmonic drive, the leadscrew mechanism provides a zero backlash and stiffer driving system.

1.6 End Effector of Robot

The end effector is connected to the main frame of the robot through the wrist. A typical wrist including three rotary axes allowing roll, pitch and yaw. Although most wrists use three rotary axes, there are applications, which require only two axes of motion. The wrist should be designed to be as light as possible. Reductions of weight at the wrist increases the maximum allowable load and reduce the moment of inertia, which improves the dynamic performance of the robot arm.

End effectors fall into two categories: grippers and tools for process applications, such as welding torches, painting guns, drills, and grinders. Grippers are used in handling, machine loading, and assembly applications. In most grippers the mechanism is actuated by a pneumatic piston, which moves the gripper fingers. When the robot is handling glass products or parts with highly polished surfaces, a vacuum type gripper can be used.

1.7 Programming of Robots

To state an algorithm, it is necessary, of course, to be able to write it down and express it logically, but it is also necessary, if it is to be executed by a machine, to state the algorithm in terms of some programming language. There are three methods used in the development of software for industrial or personal robots.

First method is the teaching method. This method consists in showing a robot what to do, with an accessory called “teaching pendant”.

The second method of programming a robot is the comprehensive method. This method is known in the industry as “world modelling” method. Instead of showing a robot what to do, this method simulates a robot procedure using three-dimensional geometric models. By simulating robot actions on a screen using geometry based on cartesian coordinates, each step can be indicated by using the model. The problem with this method is that it assumes that the robot will operate the way the model operates.

The third method consists of a robot and computer programming language. In most cases, a high-level programming language is used along with a suitable subset of motion and manipulation commands. The focus of this method is on the end-effector or manipulator, and on the instructions of what the end-effector is to do in each step of the way. There are various kinds of programming languages used in robotics.

1.8 Object of Present Investigation

The object of this investigation is to develop robotic software, which controls a 6 degree-of-freedom robotic manipulator, which has been designed and manufactured at the CAD/CAM/ROBOTICS centre of METU.

Motors of the robot will be controlled by the motion control (DELTA TAU) card. 8 motors can be controlled simultaneously with PC.

There are 6 revolute joint controlled by 6 servomotors. Generally the user specifies a path to describe the required motion of the robot in space coordinates. This information in space coordinates must be converted to joint variables. A theory must be developed to achieve this conversion and the best technique is to be chosen among the alternatives.

A graphics program will be developed to simulate this motion considering the limits on displacement, velocity and acceleration. Once a satisfactory solution is found, then the necessary information will be sent to the motion control card. The robot will then be expected to execute this motion.

The movements must be smooth, because robot can do sensitive jobs, e.g. welding or painting. In order to get the smoothness, the path must be optimized and the speed and acceleration of the motors and hand must be limited. This will be guaranteed with the software.

1.9 LITERATURE SURVEY

1.9.1 Projects with the METUROBOT

METUROBOT is the first industrial robot designed and manufactured in **CAD/CAM/ROBOTICS Centre** in Mechanical Engineering Department of

METU. There are 4 thesis studies completed on this robot. These theses are all supervised by **Prof. Dr. Bilgin Kaftanoğlu**.

The initial design of the METUROBOT was started with the thesis study "**Computer Aided Design of an Industrial Robot Arm**" **(1994)** submitted by **Erdal Çağlayan** [3]. In this study, an interactive algorithm through the use of integrated solid model, kinematic, dynamic and finite element analysis is developed. The graphical method, for visualising the dynamic performance of the designed arm, is introduced. The assembly relations of the robot are handled through a hierarchical data structure. In the study, kinematic simulation is also combined with the solid model of the complete robot assembly. The mechanical design of the transmission elements is held according to the well-known classical machine elements design procedures. In summary, this study illustrates a CAD algorithm, using advanced graphics and analysis software available, for an optimal robot arm, which can be applied to any type of robot arm by modifying the presented computer program.

The following thesis study is "**The Computer Aided Design of an Industrial Robot**" **(1997)** submitted by **Tolga Ünver** [17]. In this thesis, the preliminary design of METUROBOT is performed. In order to actuate, motors and drive systems are chosen. Taking some criteria into account, several alternatives are designed and modelled using the software Pro-Engineer. Using program facility, a simulation is performed and arm structures are modified to optimise the work envelop. Then, considering the dynamic performance of transmission systems, work envelopes obtained, and manufacturing and assembly operations, the alternatives are discussed and a final decision was made.

Next thesis study is "**Virtual Modelling, Planning and Production of Parts of an Industrial Robot**" **(1999)** submitted by **Şükrü Bülent Toker** [16]. The aim of this thesis is to construct a 6 degrees of freedom robot designed previously. The aim is to produce a stiff and no backlash robotic system. In the thesis, production of parts, assembly and initial operation of METUROBOT is performed. 3D solid modelling and virtual assembly techniques are used to minimise the problems encountered in the assembly stage. Next, some parts are modified and then, productions of parts are started. The parts are manufactured mainly using the capabilities of METU CAD/CAM/ROBOTICS Centre. Some sponsors are found for some of the components to be purchased and manufacturing in several companies. However, the production of some parts and the assembly of the robot are not completed.

Last thesis study completed with the METUROBOT is "**Production, Assembly and Application of an Industrial Robot**" **(2001)**, submitted by **Oykun Eren** [4]. Parts of this thesis are; checking of initial design and accomplishment of the design modifications where necessary, finishing of the production of parts, physical assembly to its final stage including the painting, the electrical and electronical assembly of motors with servo drives and control circuitry together with the design and construction of a control box, and finally, initial testing using joint variables.

1.9.2 Other Studies Related to the Topics in the Thesis

E. İlhan Konukseven [6] completed a thesis on graphical simulation and programming of robots, "**Graphical Simulation and Programming of Robots**" **(1989)**. In his study, he constructed a 3D graphical model of Puma type manipulator. He solved forward and inverse kinematics of the robot and generated path. The representation of the robot is wire-frame and animation is included in the thesis.

Anas Abidi [1] completed the thesis "**Man-Machine Interface Software Development for an Industrial Robot**"(2002). He developed a graphical user interface "GUI" for ABB-IRB 2000 in the CAD/CAM/ROBOTICS centre. He used solid models of the robot in the animation part of the software and he wrote off-line programs to debug on the graphical simulation and execute on the robot.

He also developed a collision detection algorithm for the parts of the robot and robot-object.

1.10 Thesis Outline

Chapter 2: Kinematic Analyses

This chapter includes basic concepts of robot kinematics. Forward and inverse kinematic analyses, i.e. position, velocity and acceleration analyses are derived. Implementations of these analyses to the computer program is also discussed.

Chapter 3: Trajectory Planning

In the trajectory planning chapter, path planning of the tip point of the METUROBOT is explained. Path optimisation, i.e. movement in minimum time, is also discussed in this chapter.

Chapter 4: Simulation Framework

In this chapter, the simulation framework system of the METUROBOT software is explained.

Chapter 5: Motion Control

This chapter explains how the METUROBOT is controlled through a PC and how it can be implemented in the thesis work.

Chapter 6: Software of “MMI of METUROBOT”

In this chapter, software of “Man-Machine Interface (MMI) of METUROBOT” is explained.

Chapter 7: Discussion & Conclusion

This chapter concludes the thesis by summarizing the work done and discussing possible future work.

Chapter 2

KINEMATIC ANALYSES

2.1 Introduction

Kinematics is the science of motion. Robot is considered as a series of links connected by joints. Joints of robots have one degrees of freedom. There are two types of joints. *Revolute* (or *rotationary*) joints provide one degree of rotation and *prismatic* joints provide one degrees of translation.

The robot user/programmer is interested in the position and orientation (pose) of the end-effector. However, the robot is controlled by the joint actuators and actuators controls the joints in terms of angles.

There are two main parts of these analyses, *forward* and *inverse*. In forward analyses, one knows the angular position, velocity and acceleration of each motor and wants to know position, velocity and acceleration of the end-effector. In inverse analyses, features of the end-effector are known and one wants to know the features of each motor. In robot programming applications, inverse kinematic analyses are useful, because programmer wants to manipulate the end-effector of the robot.

In the kinematic analyses, the translational and rotational relations between adjacent links must be described. Hartenberg & Denavit proposed a matrix method for this purpose. First HD convention parameters will be expressed and position analyses will be done accordingly.

After the position analyses, velocity and accelerations will be done. In these analyses, a special matrix, Jacobian matrix will be formed and velocity & acceleration analyses will be done.

2.2 Hartenberg-Denavit (HD) Convention

A systematic technique for establishing the displacement matrix for each two adjacent links of a mechanism was proposed by Hartenberg and Denavit in 1955. This convention will be used in this investigation [10].

The HD convention is mainly implemented in robot manipulators, which consist of an open kinematic chain in which each joint contains one degree of freedom and the joint is either revolute or prismatic. The HD convention is implemented through the following steps (Figure 2.1):

1. Number the links and joints, starting at the base. The stationary base is denoted as link *0* and the end effector is link *n*, as demonstrated in figure. Link *i* moves in respect to link *i-1* around (for revolute) or along (for prismatic) joint *i*.
2. Establish links' coordinate system for each of the joints according to the following rules:
   a) The Z_n axis is chosen along the axis of motion of joint *n+1*. For a revolute joint, link *n+1* rotates in respect to link *n* around the $+Z_n$ axis in the amount of $+\theta_{n+1}$; for a prismatic joint, link *n+1* is displaced relative to link *n* along the $+Z_n$ axis in the amount of $+d_{n+1}$.
   b) The X_{n+1} axis is chosen perpendicular to the Z_n axis (i.e. it is perpendicular to both Z_n and Z_{n+1}). If Z_{n+1} and Z_n do not intersect, then the X_{n+1} axis is along the common normal to Z_{n+1} and Z_n and

its direction is defined from Z_n toward the Z_{n+1} axis. If, however, Z_n and Z_{n+1} do intersect, the direction of X_{n+1} axis is not defined and it can be chosen in either of the two possible directions. In addition, if the Z_n and Z_{n+1} axes are collinear, the X_{n+1} axis can be chosen anywhere in the plane perpendicular to them.

c) The Y_{n+1} axis is chosen to complete a right-handed coordinate system.

Note that the assignment of coordinate system is not unique. For example, there are several possibilities for the selection of the direction of the X_{n+1} axis.

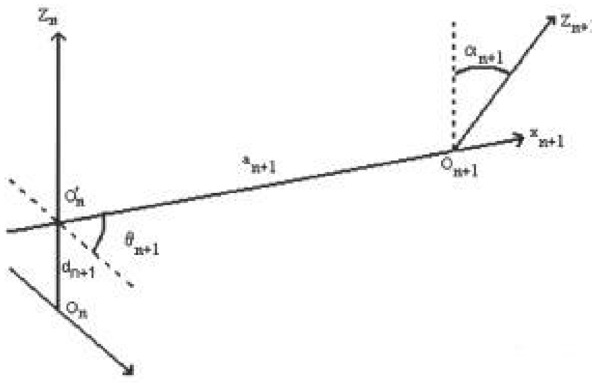


Figure 2.1 HD Convention Parameters

3. Define the joint parameters, which are the four geometric quantities θ_{n+1}, d_{n+1}, a_{n+1}, α_{n+1}.

$\boldsymbol{\theta_{n+1}}$ is the angle between the X_n and the X_{n+1} axis, obtained by rotating X_n into X_{n+1} around the Z_n axis. For a revolute joint, θ_{n+1} is a variable and for a prismatic joint θ_{n+1} is a constant parameter.

d_{n+1} is the coordinate of the origin of O_{n+1} frame on the Z_n axis i.e., the distance between the origin of O_n frame to the intersection of the Z_n axis with the X_{n+1} axis. For a prismatic joint d_{n+1} is a variable, and for a revolute joint d_{n+1} is a constant parameter.

a_{n+1} is the distance between Z_n and Z_{n+1} axis measured along the negative direction of X_{n+1} from its origin to where it intersects the Z_n axis (a constant parameter).

α_{n+1} is the angle between the Z_n axis and the Z_{n+1} axis, obtained by rotating Z_n into Z_{n+1} around the X_{n+1} axis (a constant parameter).

Using these parameters, the orientation matrix C^{i}_{i-1} of link i with respect to link i-1 is given by (Eq 2.1):

$$C^{i}_{i-1} = \begin{bmatrix} \cos(\theta_i) & -\cos(\alpha_i)\cdot\sin(\theta_i) & \sin(\alpha_i)\cdot\sin(\theta_i) \\ \sin(\theta_i) & \cos(\alpha_i)\cdot\cos(\theta_i) & -\sin(\alpha_i)\cdot\cos(\theta_i) \\ 0 & \sin(\alpha_i) & \cos(\alpha_i) \end{bmatrix} \tag{2.1}$$

2.3 HD PARAMETERS:

To find HD parameters of the robot, the wire-frame model of the robot must be drawn (Figure 2.2):

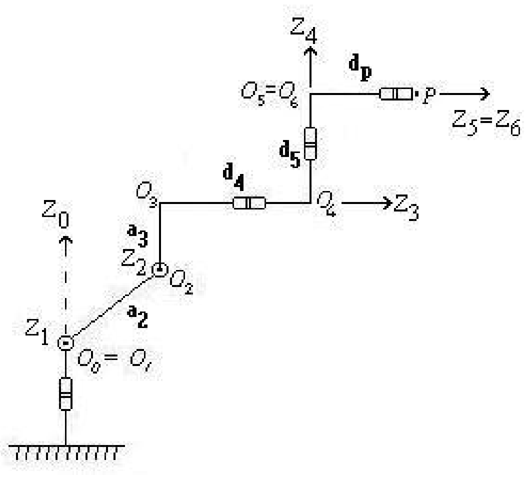


Figure 2.2 Model of METUROBOT

According to the robot link and distance definitions, the HD parameters of this robot are (Table 2.1):

Table 2.1 HD Parameters of the robot

Joint Number	a	d	α	θ
1	0	0	π/2	j.v.(Joint Variable)
2	a_2	0	0	j.v.
3	a_3	0	-π/2	j.v.
4	0	d_4	-π/2	j.v.
5	0	d_5	π/2	j.v.
6	0	0	0	j.v.

By using these parameters, the following rotation matrices can be formed between links. $C_{(i-1)(i)}$ is the rotation matrix between link (i-1) and link (i). The closed form of $C_{(i-1)(i)}$ is;

$$C = e^{\tilde{u}_3 \cdot \theta} \cdot e^{\tilde{u}_1 \cdot \alpha} \tag{2.2}$$

where,

$$\bar{u}_1 = \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix} \qquad \bar{u}_2 = \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix} \qquad \bar{u}_3 = \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}$$

and,

$$\tilde{n} = \begin{pmatrix} 0 & -n_3 & n_2 \\ n_3 & 0 & -n_1 \\ -n_2 & n_1 & 0 \end{pmatrix} \qquad \text{if} \qquad \bar{n} = \begin{pmatrix} n_1 \\ n_2 \\ n_3 \end{pmatrix}$$

therefore,

$$e^{\tilde{u}_1 \cdot \theta} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos(\theta) & -\sin(\theta) \\ 0 & \sin(\theta) & \cos(\theta) \end{pmatrix} \quad \text{and} \quad e^{\tilde{u}_3 \cdot \theta} = \begin{pmatrix} \cos(\theta) & -\sin(\theta) & 0 \\ \sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{pmatrix} \tag{2.3)\&(2.4}$$

According to the (Eq. 2.2), the rotation matrices can be formed as;

$$C_{01} = e^{\tilde{u}_3 \cdot \theta_1} \cdot e^{\tilde{u}_1 \cdot \frac{\pi}{2}} = \begin{pmatrix} \cos(\theta_1) & 0 & \sin(\theta_1) \\ \sin(\theta_1) & 0 & -\cos(\theta_1) \\ 0 & 1 & 0 \end{pmatrix} \tag{2.5}$$

$$C_{12} = e^{\tilde{u}_3 \cdot \theta_2} = \begin{pmatrix} \cos(\theta_2) & -\sin(\theta_2) & 0 \\ \sin(\theta_2) & \cos(\theta_2) & 0 \\ 0 & 0 & 1 \end{pmatrix} \tag{2.6}$$

$$C_{23} = e^{\tilde{u}_3 \cdot \theta_3} \cdot e^{\tilde{u}_1 \cdot -\frac{\pi}{2}} = \begin{pmatrix} \cos(\theta_3) & 0 & -\sin(\theta_3) \\ \sin(\theta_3) & 0 & \cos(\theta_3) \\ 0 & -1 & 0 \end{pmatrix} \tag{2.7}$$

$$C_{34} = e^{\tilde{u}_3 \cdot \theta_4} \cdot e^{\tilde{u}_1 \cdot -\frac{\pi}{2}} = \begin{pmatrix} \cos(\theta_4) & 0 & -\sin(\theta_4) \\ \sin(\theta_4) & 0 & \cos(\theta_4) \\ 0 & -1 & 0 \end{pmatrix} \tag{2.8}$$

$$C_{45} = e^{\tilde{u}_3 \cdot \theta_5} \cdot e^{\tilde{u}_1 \cdot \frac{\pi}{2}} = \begin{pmatrix} \cos(\theta_5) & 0 & \sin(\theta_5) \\ \sin(\theta_5) & 0 & -\cos(\theta_5) \\ 0 & 1 & 0 \end{pmatrix} \tag{2.9}$$

$$C_{56} = e^{\tilde{u}_3 \cdot \theta_6} = \begin{pmatrix} \cos(\theta_6) & -\sin(\theta_6) & 0 \\ \sin(\theta_6) & \cos(\theta_6) & 0 \\ 0 & 0 & 1 \end{pmatrix} \tag{2.10}$$

2.4 Position Analyses

2.4.1 Forward Position Analysis

The position and orientation of the end-effector is determined using joint angles. This is named as forward position analysis. This analysis is done symbolically. Found position and orientation elements are used in other kinematic analyses. In robotic applications, generally inverse kinematic analyses are used, because, generally the pose (position & orientation) of end-effector is known values but joint angles are unknown values. The orientation can be found first, because, part of the position is found using orientation.

2.4.1.1 Orientation of the End-Effector

Orientation of the end-effector is found by multiplying all rotation matrices, because, the lengths of the links and offsets cannot affect the orientation. The orientation matrix is then;

$$C = C_{01} \cdot C_{12} \cdot C_{23} \cdot C_{34} \cdot C_{45} \cdot C_{56} \tag{2.11}$$

In the general rotation matrices, not all the elements are independent. We can express rotation matrices with 3 independent elements. These are called Euler angles. In this thesis, Euler angles with 123 (yaw, pitch, raw) sequence will be used. Before converting this rotation matrix to the Euler angles we must define rotation matrix with yaw, pitch and roll angles. First, rotation matrices about the principal axes are evaluated.

A rotation of ψ radians about the x-axis is:

$$R_x = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos(\psi) & -\sin(\psi) \\ 0 & \sin(\psi) & \cos(\psi) \end{pmatrix} \tag{2.12}$$

A rotation of θ radians about the y axis is:

$$R_y = \begin{pmatrix} \cos(\theta) & 0 & \sin(\theta) \\ 0 & 1 & 0 \\ -\sin(\theta) & 0 & \cos(\theta) \end{pmatrix} \tag{2.13}$$

and, a rotation of ϕ radians about the y axis is:

$$R_z = \begin{pmatrix} \cos(\phi) & -\sin(\phi) & 0 \\ \sin(\phi) & \cos(\phi) & 0 \\ 0 & 0 & 1 \end{pmatrix} \tag{2.14}$$

For Euler 123 angle sequence the final rotation is the multiplication of the above matrices in the following way:

$$C = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos(\phi) & -\sin(\phi) \\ 0 & \sin(\phi) & \cos(\phi) \end{pmatrix} \cdot \begin{pmatrix} \cos(\theta) & 0 & \sin(\theta) \\ 0 & 1 & 0 \\ -\sin(\theta) & 0 & \cos(\theta) \end{pmatrix} \cdot \begin{pmatrix} \cos(\psi) & -\sin(\psi) & 0 \\ \sin(\psi) & \cos(\psi) & 0 \\ 0 & 0 & 1 \end{pmatrix} \tag{2.15}$$

$$C = \begin{pmatrix} \cos(\theta)\cdot\cos(\psi) & -\cos(\theta)\cdot\sin(\psi) & \sin(\theta) \\ \sin(\phi)\cdot\sin(\theta)\cdot\cos(\psi) + \cos(\phi)\cdot\sin(\psi) & -\sin(\phi)\cdot\sin(\theta)\cdot\sin(\psi) + \cos(\phi)\cdot\cos(\psi) & -\sin(\phi)\cdot\cos(\theta) \\ -\cos(\phi)\cdot\sin(\theta)\cdot\cos(\psi) + \sin(\phi)\cdot\sin(\psi) & \cos(\phi)\cdot\sin(\theta)\cdot\sin(\psi) + \sin(\phi)\cdot\cos(\psi) & \cos(\phi)\cdot\cos(\theta) \end{pmatrix}$$

ψ, θ and ϕ can be calculated using rotation matrix expressed above:

$$\theta = \operatorname{asin}(C_{13}) \tag{2.16}$$

$$\psi = \operatorname{atan2}(C_{11}, -C_{12}) \tag{2.17}$$

$$\phi = \operatorname{atan2}(C_{33}, -C_{23}) \tag{2.18}$$

There are two possibilities for the selection of the sign of θ, but, the sign selection is not important. The sign is selected positive in this study.

2.4.1.2 Position of the Tip Point

According to the figure 2.2, the equation of the tip point position is;

$$P = 1201\bar{u}_3 + a_2\bar{u}_1^{(2)} + a_3\bar{u}_1^{(3)} + d_4\bar{u}_3^{(3)} + d_5\bar{u}_3^{(4)} + d_p\bar{u}_3^{(6)} \tag{2.19}$$

where,
1201 is the constant distance between base and O_0.
u_1 is the unit vector in the direction of x, on base frame
u_3 is the unit vector in the direction of z, on base frame
and,
$u_1^{(x)}$ is the unit vector in the direction of x, on xth frame

$u_1^{(x)}$ is expressed in the base frame as;

$$u_1^{(X)} = C_{0X} \cdot u_1 \tag{2.20}$$

The position of O_5 with respect to O_0 is defined here as R so that;

$$P = 1201 \cdot u_3 + R + d_p \cdot C_{06} \cdot u_3 \tag{2.21}$$

where elements of R is;

$$R_1 = \left(a_3\cdot\cos\left(\theta_{23}\right) - d_4\cdot\sin\left(\theta_{23}\right) - d_5\cdot\cos\left(\theta_{23}\right)\cdot\sin\left(\theta_4\right) + a_2\cdot\cos\left(\theta_2\right)\right)\cdot\cos\left(\theta_1\right) - \sin\left(\theta_1\right)\cdot d_5\cdot\cos\left(\theta_4\right) \quad (2.22)$$

$$R_2 = \left(a_3\cdot\cos\left(\theta_{23}\right) - d_4\cdot\sin\left(\theta_{23}\right) - d_5\cdot\cos\left(\theta_{23}\right)\cdot\sin\left(\theta_4\right) + a_2\cdot\cos\left(\theta_2\right)\right)\cdot\sin\left(\theta_1\right) + \cos\left(\theta_1\right)\cdot d_5\cdot\cos\left(\theta_4\right) \quad (2.23)$$

$$R_3 = a_2\cdot\sin\left(\theta_2\right) + a_3\cdot\sin\left(\theta_{23}\right) + d_4\cdot\cos\left(\theta_{23}\right) - d_5\cdot\sin\left(\theta_{23}\right)\cdot\sin\left(\theta_4\right) \quad (2.24)$$

Then, the elements of the P matrix are;

$$P_1 = R_1 + c_{13}\cdot d_p \quad (2.25)$$

$$P_2 = R_2 + c_{23}\cdot d_p \quad (2.26)$$

$$P_3 = 1201 + R_2 + c_{33}\cdot d_p \quad (2.27)$$

where, c_{13}, c_{23}, c_{33} are the elements of C_{06} matrix.

2.4.2 Inverse Position Analysis

2.4.2.1 Introduction

Inverse position analysis is to find joint angles from given pose of the end-effector. First we must determine rotation & translation matrices with given position and orientation. For the simplicity, we convert position of tip point with respect to the base, to position of wrist point with respect to O_0. Then the inverse position analysis is done using the wrist point position.

After the matrices are formed, using the detailed expressions of the elements, we can find the joint variables of the robot. The elements in the position matrix are independent, but in rotation matrix, only 3 of 9 elements are independent. This means, there are 6 independent equation for 6 unknown joint variables.

Generally, the inverse position analysis is solved with fully analytical method, because, generally translational elements are consisting 3 joint variables. With 3 equation and 3 unknowns, joint variables can be solved. However, METUROBOT has special design and the translation matrix, that is, Rx, Ry and Rz are functions of 4 joint variables. Rotation matrix is a function of all joint variables. Because of this, the semi-analytical method is used. In this method, we treat one of the joint variables (θ_1) as if it is known. Then we solve θ_2, θ_3, θ_4, depending on θ_1, using translation matrix equations. Then, there is more than one approach to proceed from this point on. One approach is to find θ_4, θ_5, θ_6 also in terms of θ1, using rotational part of equations, in addition to the previously found θ2, θ3 and θ4. Now, there are two different θ4 expressions in terms of θ1. The last step is to equate these two expressions of θ4 in order to extract θ1 out of them [10].

The second way of solution is to find θ1 using one element of the rotation matrix, which is dependent only on the first four joint variables. This procedure is taken from the paper written by Balkan, Özgören, Arıkan and Baykurt [2]. Its details are explained in section 2.4.2.2.

In the first approach, all three sign ambiguities appear explicitly, but only two of them appear explicitly in the second one. In other words, the first approach gives all multiple solutions, whereas the second approach gives only two of the solutions. Therefore, in the second approach, all of the singular and multiple configurations may not be seen clearly. However, due to a bug in programming the first approach, it hasn't been so far possible to use it effectively. Therefore, the second approach is preferred in this thesis.

2.4.2.2 Formulation

The first thing to do is to convert tip point position to wrist point position. From (Eqs 2.25-2.27);

$$R_1 = P_1 - c_{13} \cdot d_p \quad (2.28)$$

$$R_2 = P_2 - c_{23} \cdot d_p \quad (2.29)$$

$$R_3 = P_3 - c_{33} \cdot d_p - 1201 \quad (2.30)$$

We start with 3 translational elements of wrist point position and treating as if θ_1 is known (from eqs 2.22-2.24),

$$R_1 = A_1 \cdot \cos(\theta_1) - A_2 \cdot \sin(\theta_1) \quad (2.31)$$

$$R_2 = A_1 \cdot \sin(\theta_1) + A_2 \cdot \cos(\theta_1) \quad (2.32)$$

where,

$$A_1 = a_3 \cdot \cos(\theta_{23}) - d_4 \cdot \sin(\theta_{23}) - d_5 \cdot \cos(\theta_{23}) \cdot \sin(\theta_4) + a_2 \cdot \cos(\theta_2) \quad (2.33)$$

$$A_2 = d_5 \cdot \cos(\theta_4) \quad (2.34)$$

From the equations (2.31 & 2.32), we get;

$$A_1 = R_1 \cdot \cos(\theta_1) + R_2 \cdot \sin(\theta_1) \quad (2.35)$$

$$A_2 = R_2 \cdot \cos(\theta_1) - R_1 \cdot \sin(\theta_1) \quad (2.36)$$

θ_4 can be found from eq. (2.34);

$$\theta_4 = \sigma_4 \cdot \operatorname{acos}\left(\frac{A_2}{d_5}\right) \qquad \sigma_4 \text{ is either (-) or (+)} \qquad (2.37)$$

In the home position, the initial value of θ_4 is –90 degrees. When the robot is requested to be in right armed configuration, θ_4 must be small angle, and σ_4 is selected positive, otherwise, σ_4 is selected negative. For details, see 2.4.2.4

from the equations (2.24 & 2.33);

$$\cos(\theta_2) = \frac{A_1 - \left(a_3 \cdot \cos(\theta_{23}) - d_4 \cdot \sin(\theta_{23}) - d_5 \cdot \cos(\theta_{23}) \cdot \sin(\theta_4)\right)}{a_2} \qquad (2.38)$$

and,

$$\sin(\theta_2) = \frac{R_3 - \left(a_3 \cdot \sin(\theta_{23}) + d_4 \cdot \cos(\theta_{23}) - d_5 \cdot \sin(\theta_{23}) \cdot \sin(\theta_4)\right)}{a_2} \qquad (2.39)$$

also there is an equation,

$$\cos(\theta_2)^2 + \sin(\theta_2)^2 = 1 \qquad (2.40)$$

all these three equations combined and result is,

$$0 = B_2 \cdot \sin(\theta_{23}) + A_2 \cdot \cos(\theta_{23}) + C_2 \qquad (2.41)$$

where,

$$A_2 = 2 \cdot A_1 \cdot d_5 \cdot \sin(\theta_4) - 2 \cdot R_3 \cdot d_4 - 2 \cdot A_1 \cdot a_3 \qquad (2.42)$$

$$B_2 = 2 \cdot A_1 \cdot d_4 + 2 \cdot R_3 \cdot d_5 \cdot \sin(\theta_4) - 2 \cdot R_3 \cdot a_3 \qquad (2.43)$$

$$C_2 = A_1^2 + R_3^2 - d_5^2 \cdot \cos(\theta_4)^2 + d_5^2 + d_4^2 + a_3^2 - a_2^2 - 2 \cdot a_3 \cdot d_5 \cdot \sin(\theta_4) \qquad (2.44)$$

the solution of this equation is,

$$\theta_{23} = 2\cdot \operatorname{atan}(t) \tag{2.45}$$

where,

$$t = \frac{B_2 + \sigma_2\sqrt{B_2^{\,2} + A_2^{\,2} - C_2^{\,2}}}{A_2 + C_2} \qquad \sigma_2 \text{ is either (-) or (+)} \tag{2.46}$$

When $\sigma2$ is positive, the robot is in elbow up configuration, when $\sigma2$ is negative, the robot is in elbow down configuration. For details, see 2.4.2.4.

We found θ_{23}, which is $\theta_2 + \theta_3$. We can find θ_2 from the above equations (2.38 & 2.39) as,

$$\theta_2 = \operatorname{atan2}\left(\cos\left(\theta_2\right), \sin\left(\theta_2\right)\right) \tag{2.47}$$

Now, θ_2, θ_3, θ_4 is found in terms of θ_1. To find true θ_1, the rotation matrix C must be used. From the (Eq. 2.11),

$$\left(C_{01}\cdot C_{12}\cdot C_{23}\cdot C_{34}\right)^{-1}\cdot C = C_{45}\cdot C_{56} = B \tag{2.48}$$

When we manipulate the matrix calculations and we equate the last elements, the equality is;

$$\left(-c_{13}\cdot c(\theta_4) - c_{23}\cdot s(\theta_4)\cdot c(\theta_{23})\right)\cdot s(\theta_1) + \left(c_{23}\cdot c(\theta_4) - c_{13}\cdot s(\theta_4)\cdot c(\theta_{23})\right)\cdot c(\theta_1) + \left(-s(\theta_4)\cdot s(\theta_{23})\cdot c_{33}\right) = 0 \tag{2.49}$$

The joint variables found before are the functions of θ_1, then above equation becomes the function of θ_1. The true θ_1 can be found by searching it through the range. In this search algorithm, inverse interpolation method is used. This method is explained in the following section. After finding true θ_1 to θ_4, we can find θ_5 and θ_6 from equality of other elements of eq.2.49.

$$\sin(\theta_5) = b_{13} \tag{2.50}$$

$$-\cos(\theta_5) = b_{23} \tag{2.51}$$

$$\sin(\theta_6) = b_{31} \tag{2.52}$$

$$\cos(\theta_6) = b_{32} \tag{2.53}$$

then,

$$\theta_5 = \operatorname{atan2}(-b_{23}, b_{13}) \tag{2.54}$$

$$\theta_6 = \operatorname{atan2}(b_{32}, b_{31}) \tag{2.55}$$

2.4.2.3 Inverse Interpolation

When the function known at discrete points, the function values of the interior points can be found by interpolation. Finding the argument value for the given value of the function is known as inverse interpolation. Therefore, the roots of the given function known at discrete points can be found with inverse interpolation.

When the given discrete points of the function are;

$f(x_0) = y_0$

$f(x_1) = y_1$

$f(x_2) = y_2$

The desired value of the function is y (y0<y<y2), the argument for the desired value (the x value for the desired y) can be calculated as;

$$x = \frac{(y - y_1)\cdot(y - y_2)}{(y_0 - y_1)\cdot(y_0 - y_2)}\cdot x_0 + \frac{(y - y_0)\cdot(y - y_2)}{(y_1 - y_0)\cdot(y_1 - y_2)}\cdot x_1 + \frac{(y - y_0)\cdot(y - y_1)}{(y_2 - y_0)\cdot(y_2 - y_1)}\cdot x_2 \qquad (2.56)$$

There is an example for the inverse interpolation below;
The values of the discrete function are;

$f(0) = -30$

$f(5) = -15$

$f(10) = 80$

When we want to find the approximate root of the function (f(x) = 0 , i.e. y = 0), the calculation gives, x = 8.852.

2.4.2.4 Configuration Selection and Singular Points

There are multiple configurations due to the sign ambiguities in the solution of the inverse position. There are three different sign ambiguities and these ambiguities cause three configurations to select. These are;

- Elbow up/down configuration,
- Wrist up/down configuration, and
- Right/left armed configuration.

The selection of right/left armed configuration is allowed by the MMI software. This ambiguity (sign selection of σ_4) is shown in the eq. 2.37. The other configurations are defined by the designer of the METUROBOT. Due to the construction and physical limits, elbow up and wrist up configurations are used in the system. By selecting σ_2 positive, the elbow up configuration is chosen. (eq. 2.46). There is no wrist configuration selection in the equations, the wrist up configuration is selected with elbow up configuration and selected arm configuration.

There are also some singular points in the system. First singular configuration is observed when the point O_6 is on the axis of first joint Z_0. In this configuration, θ_1 can have arbitrary value. The second singular configuration is observed when O_6 is coincident with O_1. When this occurs, the θ_{32} become arbitrary. This two type of singularity cannot be encountered in the real cases, because these configurations cannot be reached by the real robot.

The last and most important type of singularity in the METUROBOT is the singularity in the wrist. When $\theta_5=0$ or $\theta_5=\pm180$ degrees, the θ_4 and θ_5 cannot be solved separately, but the value of $\theta4+\theta5$ can be determined.

2.5 VELOCITY AND ACCELERATION ANALYSES

2.5.1 JACOBIAN MATRICES:

Before doing velocity analyses, jacobian matrix must be clarified. Mainly it is the matrix is the matrix between task space velocities and joint space velocities.

We can find J_P and J_A vectors by using C (rotation) matrix found in forward position analysis. P matrix is the tip point position matrix. When the Jacobian matrix is found, it is used for both velocity and acceleration analyses.[11]

$$J_P = \begin{pmatrix} J_{P1} & J_{P2} & J_{P3} & J_{P4} & J_{P5} & J_{P6} \\ J_{A1} & J_{A2} & J_{A3} & J_{A4} & J_{A5} & J_{A6} \end{pmatrix} \tag{2.57}$$

$$\text{where,} \quad J_{Pn} = \frac{\delta}{\delta q_n} P$$

$$J_{An} = \mathrm{col}\left[\left(\frac{\delta}{\delta q_n} C\right) \cdot C^T\right]$$

The Jacobian matrix can be found for wrist point too, but user wants to control the movement of end-effector, then we must construct the tip point Jacobian matrix. The position matrix of the tip point was:

$$P = \begin{pmatrix} R_1 + C_{13} \cdot d_p \\ R_2 + C_{23} \cdot d_p \\ R_3 + C_{33} \cdot d_p + 1201 \end{pmatrix}$$

P_1, P_2, P_3 and C_{13}, C_{23}, C_{33} are defined in the forward position analyses.

Full Jacobian matrix is very large but its construction is straightforward. The usage of Jacobian method is time consuming when doing by hand, but its time saving method when doing with a computer program. The elements of the Jacobian matrix and its derivative are on the Appendix B.

2.5.2 FORWARD and INVERSE VELOCITY ANALYSES

For the given angular position and velocities of the joints, i.e. motors, the velocity of the wrist and tip point can be calculated easily with the help of jacobian matrices. The inverse of this is also very simple, when this matrix is constructed.

Elements of jacobian matrix are quite lengthy, but a computer programmer can write a function to calculate a jacobian matrix, and its inverse, with an input of angular position of the motors. When this matrix is calculated, one can easily evaluate tip point velocity using motor angular velocities or vice versa.

$$\begin{pmatrix} V_P \\ \omega_P \end{pmatrix} = J_P \cdot \begin{pmatrix} \dot{q}_1 \\ \dot{q}_2 \\ \dot{q}_3 \\ \dot{q}_4 \\ \dot{q}_5 \\ \dot{q}_6 \end{pmatrix} = J_P \cdot \dot{Q} \quad \text{where,} \tag{2.58}$$

V_P is the tip poin t linear velocity vector and

ω_P is the tip point angular velocity vector.

q_x is the angular position of the x_{th} joint

J_p is a 6x6 jacobian matrix, whose elements are dependent on joint variables, forward and inverse velocity analyses can be done using normal and inverse of J_p.

$$V = J_P \cdot \dot{Q} \tag{2.59}$$

$$\dot{Q} = J_P^{-1} \cdot V \tag{2.60}$$

2.5.3 FORWARD and INVERSE ACCELERATION ANALYSES

For the given angular position, velocity, and acceleration values of the joints, the acceleration of the tip point can be calculated. For this, partial derivative of the jacobian matrix, with respect to the joint variables, is used.

Main acceleration equation is;

$$J_p \cdot \ddot{Q} + \dot{J}_p \cdot \dot{Q} = \dot{V} \tag{2.61}$$

then,

$$\ddot{Q} = J_p^{-1} \cdot \left(\dot{V} - \dot{J}_p \cdot \dot{Q} \right) \tag{2.62}$$

Q, J_P and $\dot{J}_P$ are found from the velocity analyses and used for forward or inverse acceleration analyses.

CHAPTER 3

TRAJECTORY PLANNING

3.1 Introduction

In most of the robotic applications, it is necessary for the manipulator to follow the planned path. In some applications, such as, painting and welding, the tip point of the robot, that is hand of the robot, needs to be more sensitive in motion. In all applications, there are via points to touch or to pass. Sometimes it is necessary for the operation, or sometimes they are via points for avoiding obstacles.

These via points, starting point and ending point are specified in general task space coordinates. But, how the robot will move between these points, is the problem of trajectory planning. For example, the operator gives two points in the task space. There are 2 choices in general, trajectory planning in the task space or planning in the joint space. Planning in the task space can be seen more smooth. But, for the robot, to go on a circular path is much easier than to go on a straight line, because, the robot has revolute joints. And also the joint trajectories are easier to plan.

The other decision, which must be made, is whether the trajectory planning should take place on-line or off-line. The on-line method has the advantage of allowing the robot to respond external effects, which may cause it to modify its path. But, the curve-fitting calculations are lengthy and it limits the number of via points and decreases the accuracy of the trajectory.

The off-line method allows more way points and they can be specified more closely in time. Since most robotic applications involve repetitive operations, this method reduces the amount of computing time of trajectory planning. All the data for an application is calculated at once.

3.2 General Considerations

For the trajectory planning in the joint space, the time history of all joint variables and their first two derivatives are planned to describe the motion. In general approach, the trajectory function is updated for every interval. The function must be smooth. To guarantee this, first and second time derivative of the trajectory function must be continuos over the whole path. This method is mainly taken from the study of Dr. Konukseven [6]

If there are (n+1) points to generate trajectory, there must be (n) functions.

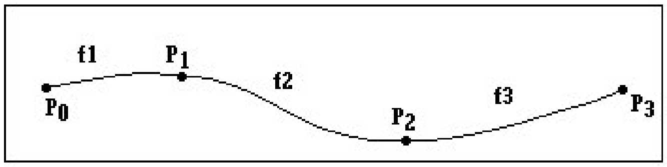


Figure 3.1 Path generation for 3 point

User wants to specify position, velocity and acceleration of the start and end points. This means 3 condition for start and 3 conditions for end points. For the intermediate points, generally the position is enough. The function values must be equal to the given condition. This gives extra (2n-2) conditions. First and second derivatives of the function at the interior nodes are equal, this also gives (n-1)+(n-1) conditions. There are 6+2n-2+2n-2=4n+2 conditions total. For example, if there are 4 points (n=3), there are 3 functions and 14 conditions. We can fit 4th degree polynomial for f_1, 3rd degree for f_2 and 4th degree for f_3. (4-3-4). Total of 5+4+5=14 constants. Or we can fit two cubic and one quantic (3-5-3).

If the number of interval increases, again there are 4n+2 conditions. To formulate this trajectory easily and independent of number of intervals, we must use (4-3-4) trajectory. If we increase the number of points, every extra interval gives +4 condition, and a function. When we add extra cubic polynomial for this extra interval, there will be no problem. (4+3+...+3+4 trajectory)

If the number of points is only two, the spline to be fitted is 5th order polynomial and it is not used in robotics generally.

3.3 4-3-4 Trajectory

When we generate a path for the robot, we generate a trajectory for each of the joints of the robot. Before generating a path, we must calculate position, velocity and accelerations of the starting and ending points, and positions of via points.

Suppose there are n points to be passed. The equation of the spline between first and second point, i.e. P_0 and P_1, is 4th order polynomial. This is:

$$F_1(t)=A_1 + A_2x + A_3 x^2 + A_4x^3 + A_5x^4 \tag{3.1}$$

The equation between last and previous points, P_{n-1} and P_n, is 4th order as well.

$$F_{n-1}(t)=Z_1 + Z_2x + Z_3 x^2 + Z_4x^3 + Z_5x^4 \tag{3.2}$$

The equation of the spline between two intermediate points, Pk and Pk+1, is:

$F_k(t)=B_1 + B_2x + B_3\,x^2 + B_4x^3$ (3.3)

Suppose end of time for 1st interval is t_1, the conditions are:

Positions are given:

$F_1(0)=P_1$, (3.4)

$F_1(t_1)=P_2$, (3.5)

$F_2(0)=P_2$, (3.6)

…

$F_{n-1}(t_{n-1})=P_n$ (3.7)

Velocities and accelerations given in the end points and equal for both interval at the intermediate points:

$F_1'(0)=P_1'$ (3.8)

$F_1''(0)=P_1''$ (3.9)

$F_{n-1}'(t_n)=P_n'$ (3.10)

$F_{n-1}''(t_n)=P_n''$ (3.11)

and,

$F_1'(t_1)=F_2'(0)$ (3.12)

$F_1''(t_1)=F_2''(0)$, (3.13)

…

$F'_{n-2}(t_{n-2})=F'_{n-1}(t_{n-1})$ (3.14)

$F''_{n-2}(t_{n-2})=F''_{n-1}(t_{n-1})$. (3.15)

When we equate the equations (3.1) and (3.4),

$P_1=A_1$ (3.16)

similarly,

$P_1'=A_2$ from (3.1) and (3.8) (3.17)

$P_1''=2.A_3$ from (3.1) and (3.9) (3.18)

$P_2=B_1$ from (3.3) and (3.6) (3.19)

$P_2=A_5.t_1^4+A_4.t_1^3+A_3.t_1^2+A_2.t_1+A_1$ from (3.1) and (3.5) (3.20)

$4.A_5.t_1^3+3.A_4.t_1^2+2.A_3.t_1+A_2=B_2$ from (3.12) (3.21)

and,

$12.A_5.t_1^2+6.A_4.t_1+2.A_3=2.B_3$ from (3.13) (3.22)

...

when we continue to the end, the following matrix is come out:

$M \text{ x } C_f = V$

Where, M is the transformation matrix, C_f is the coefficient matrix, and V is the desired values matrix. M is (3n+1)x(3n+1) matrix, where n is the number of points.

3.4 Implementation

For the 3 point (2 interval) path generation (Figure 3.2) the size of M matrix is 10x10. the points and positions with the times are shown below. Using the above (3.16 to 3.22) equations, the M x C_f = V matrices (3.23) are as following;

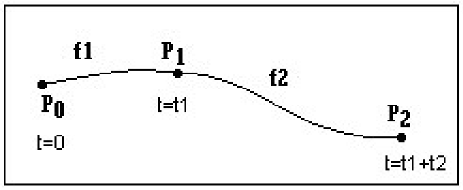


Figure 3.2 Path generation for 2 point

$$\begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & t_1 & t_1^2 & t_1^3 & t_1^4 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 2\cdot t_1 & 3\cdot t_1^2 & 4\cdot t_1^3 & 0 & -1 & 0 & 0 & 0 \\ 0 & 0 & 2 & 6\cdot t_1 & 12\cdot t_1^2 & 0 & 0 & -2 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & t_2 & t_2^2 & t_2^3 & t_2^4 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 2\cdot t_2 & 3\cdot t_2^2 & 4\cdot t_2^3 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 2 & 6\cdot t_2 & 12\cdot t_2^2 \end{pmatrix} \cdot \begin{pmatrix} A_1 \\ A_2 \\ A_3 \\ A_4 \\ A_5 \\ B_1 \\ B_2 \\ B_3 \\ B_4 \\ B_5 \end{pmatrix} = \begin{pmatrix} P_1 \\ P_1 \\ P_1 \\ P_2 \\ 0 \\ 0 \\ P_2 \\ P_3 \\ P_3 \\ P_3 \end{pmatrix}$$

Eq 3.23 MxC_f=V matrices for 3 point trajectory planning

The elements of M and V are known, then when we take inverse of the M matrix and multiply it with V matrix, the C_f matrix came out. The elements of C_f matrix are the constants of the 2 polynomials between P_0 & P_1 and P_1 & P_2.

There is an example for the trajectory planning for the given data. There are 5 points on the path, and end point velocity of the given joint is not zero, but "-4" degrees per second. Starting point velocity, starting and end point accelerations are all zero. The desired values of the positions with time are given below.

Table 3.1 Example Data for Trajectory Planning

t (sec)	angle(degree)
0	20,336
3	11,038
5	14,208
11	7,585
14	4,014

There are 5 points on the path and 4 polynomials must be fitted. We named these polynomials as $F_1(x)$, $F_2(x)$, $F_3(x)$, $F_4(x)$. Two exterior polynomials are 4th degree and two interior ones are 3rd degree. After matrix operations, these polynomials are calculated as,

$$F_1(x) := 0.35493 - 0.0193x^3 + 0.00442x^4$$

$$F_2(x) := 0.19265 - 0.04329x + 0.06524x^2 - 0.01488x^3$$

$$F_3(x) := 0.24798 + 0.03909x - 0.02405x^2 + 0.00238x^3$$

$$F_4(x) := 0.13238 + 0.008x + 0.01892x^2 - 0.01697x^3 + 0.00247x^4$$

These polynomials are combined properly, and the trajectory is found. The expression for combining the parts of the trajectories is,

$$F(x) := \left| \begin{array}{l} F_1(x) \;\; \text{if} \;\; (x > 0)\cdot(x < 3) \\ F_2(x-3) \;\; \text{if} \;\; (x > 3)\cdot(x < 5) \\ F_3(x-5) \;\; \text{if} \;\; (x > 5)\cdot(x < 11) \\ F_4(x-11) \;\; \text{if} \;\; (x > 11)\cdot(x < 14) \end{array} \right.$$

The graphs of the F(x) function and its derivatives are in the following. The first derivative is the velocity graph for the joint and the second derivative is the acceleration of the joint.

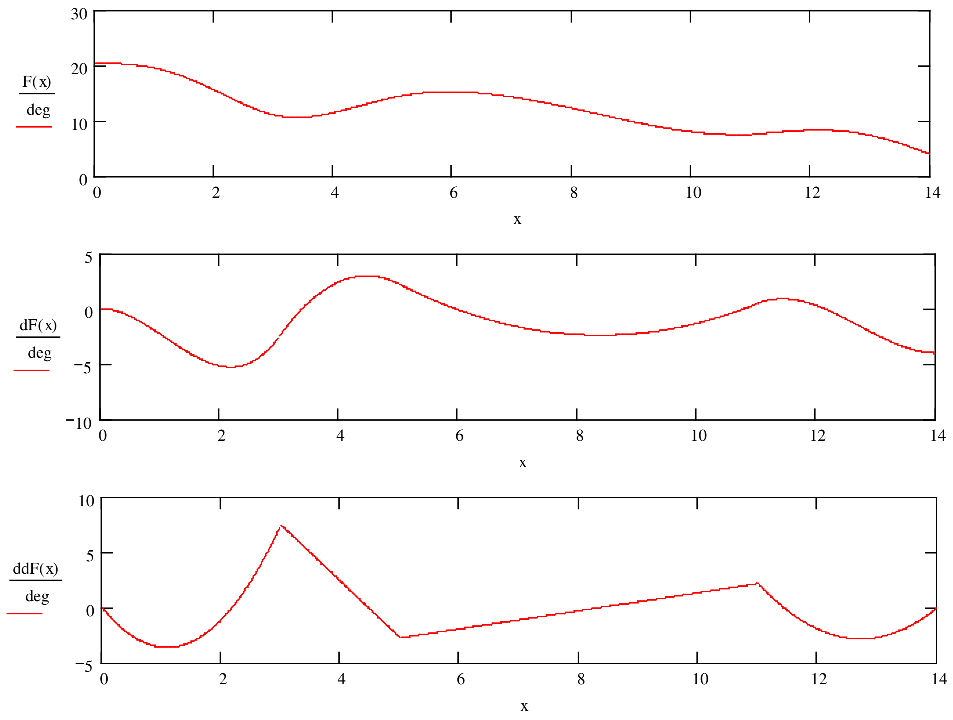


Figure 3.3 Graphs of position, velocity and acceleration of the trajectory for the example data.

3.5 Application to the Thesis

Path planning of the robot means, arrange velocities and accelerations of the joint variables such that the tip point of the robot passes from the desired points. To do this, first we calculate the inverse position analyses of the start, end desired positions of the tip point. This gives us the desired joint variables along the path.

Trajectory of each joint is calculated with the formulation above. Using the boundaries for joint velocities and accelerations for each joint, the time required to do the movement is calculated. With this, the trajectory will be found. The trajectory will be smooth enough, because, we guarantee the smoothness of all the joints.

3.6 Conclusion

In the trajectory planning, the most important thing is the smoothness of the path, because, if the path of the robot makes robot tilt or crash, or if the robot makes dangerous movements, there is no meaning to make trajectory planning or path generation. In the method used, the smoothness is the key point.

The optimisation is not considered in the trajectory planning. In fact, some optimisation criteria may be considered, such as, minimum time or minimum energy. But in the thesis, besides starting and ending points, via points are determined and the robot has to be passed from these points. This criterion avoids us to apply minimum energy optimisation. In this trajectory planning method, time must be given as the input. To optimise it, the iterative method must be used. Time optimisation in this method is also very lengthy subject for this trajectory planning method and it can be the future work of this software.

If there are two points, above formulation does not work. Because, in this case, 5th order polynomial must be fitted. Because there are 6 variables (position, velocity & acceleration of both points) to be fitted. The stability of the polynomial decreases when the value of degree of the polynomial increases. For this purpose, when two points given, software determines one via point for the path, and fits 4-4 polynomial to these 3 points.

Chapter 4

SIMULATION FRAMEWORK

4.1 Introduction

In this chapter, the graphical simulation framework of the computer program will be described. This simulation framework will be used in any particular simulation of the robot and its environment.

This chapter includes the theory of OpenGL and its implementation to the computer program of Robot Control.

4.2 OpenGL

OpenGL is the abbreviation of the Open Graphics Library for the C++. It is the Application Programming Interface (API) for graphical applications. It was created in 1992 by Silicon Graphics (SGI). This interface consists of about 150 distinct commands that user can use to specify the objects and operations needed to produce interactive 3D applications. [8]

However, OpenGL doesn't provide high-level commands for describing models of 3D objects. The geometry of OpenGL is based on vertices. The programmer inputs a command, and OpenGL draws a primitive (point, line or polygon) defined by vertices appropriate to the command. OpenGL internalises the data and functions necessary to draw the figure, rather than the programmer having to do it manually.

OpenGL routines simplify the development of graphics software, from rendering a simple geometric point, line, or filled polygon to the creation of the most complex lighted and texture-mapped curved surface. OpenGL gives access to geometric and image primitives, display lists, modeling transformations, lighting and texturing, blending and many other features. OpenGL simplifies the math needed for graphics, allowing focus on design rather than implementation. Besides, OpenGL is easy to learn, powerful and well-documented. For these reasons, OpenGL was chosen to be the building block of the visual simulation framework.

4.3 Using OpenGL in Programming Languages

To use OpenGL functions in one of the programming languages, some special libraries must be included into the code. OpenGL is developed mainly for C++ based languages. But OpenGL can be modified to the other languages. Anas Abidi [1] used OpenGL in the Visual Basic. [15]

4.3.1 Libraries used for OpenGL

Libraries and files are different for different versions of programming languages. But, the common things in implementing OpenGL are, initialising a window for 3D objects, initialising lighting, colour and surround, importing models into the window. After these, rotating and translating objects and adjusting viewing properties are done.

For using OpenGL, the libraries “gl.h” (Graphichs Library), “glu.h” (Graphics Library Utility) and “glut.h” (Graphics Library Utility Toolkit) must be included in the code. The libraries can be changed for the different versions of the programming languages, but they are free libraries and they can be found in the websites of the producer's.

4.3.2 Panel for OpenGL

For C and C++, a function is enough to initialise a window for 3D models. But, in a visual programming languages, such as Visual C++ or Borland C++ Builder, an activex control must be used. There are many GL panels on the web, and they are free to use. They are installed external packages and they are different from each other. After installing one of the panel components, the initialisation can be done. In all of the OpenGL panel components, there is an initialisation event. In this project, panel named "OpenGLAPPanel" is used. [9]

After window initialisation, the adjustment of lighting, and viewing properties are common for all panels. The commands of the initialisation used in the software of METUROBOT are in the "Implementation" section.

4.3.3 Importing Models

In the following step, the 3D models must be imported into the panel. For importing, first, we must have solid models of the parts. These solid models are not only 3D drawings of the links and objects, but they are real 3D models and they have to be prepared in AutoCAD or ProEngineer. The .stl (StereoLitography) format is very convenient because in this format, solid models are defined with triangles and triangles are easy to implement in OpenGL. But the .stl file cannot be used directly. The .stl format must be converted to .raw file format, which can be opened like .txt files and the constructing of OpenGL drawings will be very easy. After reading these vertices of the triangles, i.e. the model, the translation and rotation is applied to the model. Then, the model is ready to drawn. The example code is in the "Implementation" section.

4.3.4 Viewing Properties

The viewing properties of the window, i.e. rotate, zoom or pan of the viewport can be adjusted whenever the user wants. These are also very simple commands. All these functions are common in all OpenGL environment. There is a book called "OpenGL Programming Guide", which is known as "Red Book" of OpenGL, and this reference contains all of the information about OpenGl drawings and applications. The .pdf format of this book can be found in websites also.

4.4 Implementation of OpenGLAPPanel in MMI

In this study, an activex control named "OpenGLAPPanel" is used. This is an activex control written for Borland C++ 4.0 and 5.0. The panel component and some examples are downloaded from internet and its free of charge.

After installing panel component, we put it on the main frame. For the initialisation the following code is written on the "paint" event of the panel;

```
GLfloat Ambient[] = { 0.2f, 0.2f, 0.2f, 1.0f };
GLfloat Diffuse[] = { 0.8f, 0.8f, 0.8f, 1.0f };
GLfloat Specular[] = { 0.2f, 0.2f, 0.2f, 1.0f };
GLfloat SpecularExp[] = { 50 };
GLfloat Emission[] = { 0.1f, 0.1f, 0.1f, 1.0f };
glMaterialfv(GL_FRONT, GL_AMBIENT, Ambient);
glMaterialfv(GL_FRONT, GL_DIFFUSE, Diffuse);
glMaterialfv(GL_FRONT, GL_SPECULAR, Specular);
glMaterialfv(GL_FRONT, GL_SHININESS, SpecularExp);
glMaterialfv(GL_FRONT, GL_EMISSION, Emission);
glMaterialfv(GL_BACK, GL_AMBIENT, Ambient);
glMaterialfv(GL_BACK, GL_DIFFUSE, Diffuse);
glMaterialfv(GL_BACK, GL_SPECULAR, Specular);
glMaterialfv(GL_BACK, GL_SHININESS, SpecularExp);
glMaterialfv(GL_BACK, GL_EMISSION, Emission);
glColorMaterial(GL_FRONT_AND_BACK, GL_DIFFUSE);
glEnable(GL_COLOR_MATERIAL);
GLfloat light_position[] = { 1.0, 1.0, 1.0, 0.0 };
glLightfv(GL_LIGHT0, GL_POSITION, light_position);
glEnable(GL_LIGHT0);
glEnable(GL_LIGHTING);
glLightModeli(GL_LIGHT_MODEL_TWO_SIDE, GL_TRUE);
glDepthFunc(GL_LEQUAL);
glEnable(GL_DEPTH_TEST);
glShadeModel(GL_SMOOTH);
glClearColor(0,0,0.5,0);
glClearColor(0.4392, 0.5020, 0.5647,1.0);
glEnable(GL_CULL_FACE);
glCullFace(GL_BACK);
glEnable(GL_NORMALIZE);
glHint(GL_PERSPECTIVE_CORRECTION_HINT,GL_NICEST);
glMatrixMode(GL_PROJECTION);
glLoadIdentity();
glFrustum(-0.004,0.004,-0.004,0.004,.01,100.0);
glMatrixMode(GL_MODELVIEW);
Draw_Robot();
```

Figure 4.1 OpenGL initialisation code

Draw_Robot() is the function for importing robot links and rotating links as wanted. The part of the software, which rotates and translates links are in the following. These lines are in the function Draw_Robot().

```
double oglm[16];
glTranslatef(x_move, y_move, 0);
//Color, rotation and translation parameters for Floor
glColor3d(0.2745, 0.5098,0.70588);
glRotatef(90,1,0,0);
glPushMatrix();
Draw_Link("links//floor.raw",1);
glGetDoublev(GL_MODELVIEW_MATRIX,oglm);
//OGLtoMV(Rfloor,Tfloor,oglm);
glPopMatrix();
glRotatef(-90,1,0,0);
//Color, rotation and translation parameters for Base part of the robot
glColor3d(0.9411764,1.0,1.0);
glRotatef(90,0,0,1);
glPushMatrix();
Draw_Link("links//base.raw",2);
glGetDoublev(GL_MODELVIEW_MATRIX,oglm);
glPopMatrix();
glRotatef(-90,0,0,1);
//Color, rotation and translation parameters for Link1
glTranslatef(0,0.765,0);
glRotatef(90,0,0,1);
glRotatef(teta1,1,0,0);
glPushMatrix();
Draw_Link("links//link1.raw",3);
glGetDoublev(GL_MODELVIEW_MATRIX,oglm);
glPopMatrix();
```

Figure 4.2 Code of translation, rotation and colouring of links

This code rotates, translates the links and sets the colour of the links. İmporting links are done by the function Draw_Link(). This is another function.

```
        FILE *fp = fopen(rawfile,"r");
         if (fp == NULL)
         {
          fprintf(stderr,"Model Constructor: Couldn't open %s\n",rawfile);
          exit(-1);
         }

ntris[n]=0;
         double tmp;
         while ((fscanf(fp,"%lf %lf %lf %lf %lf %lf %lf %lf %lf \n", &tmp, &tmp, &tmp, &tmp,
            &tmp, &tmp,&tmp,&tmp,&tmp)==9))
         {
         ntris[n]++;
         }
         fclose(fp);
        ................................
         glBegin(GL_TRIANGLES);
          for (int i = 0; i < ntris[n]; i++)
          {
           glNormal3dv(tri[n][i][0]);
           glVertex3dv(tri[n][i][1]);
           glVertex3dv(tri[n][i][2]);
           glVertex3dv(tri[n][i][3]);
          }
         glEnd();
```

Figure 4.3 Code of importing links from raw file and drawing to the panel

Above code reads the triangles from the .raw files and draws the links to the panel. As mentioned before, all solid 3D models can be converted to triangles. For example, for the base of the robot, there are about 1500 triangles, i.e. the base is composed of 1500 triangles. All these trianlges are grouped together and drawn to the panel as one object. Therefore, translation and rotation of “objects” can be done very easily.

Data files, i.e. raw files are easy to handle, below, some portion of base.raw data file. In every line there are nine numbers, they are x, y and z components of three vertices of the triangles. The number of lines is the number of triangles.

```
...
0 -333.098 35.6379  0 333.098 35.6379  0 334.519 17.8835
40 334.519 17.8835  0 334.519 17.8835  0 333.098 35.6379
0 -334.519 17.8835  0 -333.098 35.6379  0 334.519 17.8835
40 334.519 17.8835  40 335 0  0 334.519 17.8835
0 -330.736 53.273  0 330.736 53.273  0 333.098 35.6379
40 333.098 35.6379  0 333.098 35.6379  0 330.736 53.273
...
```

Figure 4.4 Some portion of the base.raw file

4.5 Conclusion

Simulation framework is the one of the main parts of the thesis, because the robot and its environment is seen by the simulation. If there is an error in the generation of the path, or something else, it does not make big problem.

OpenGL is very suitable for applications like in this thesis, because it is easy to apply. For 3D solid models, its even simpler, because, 3D models in

Autocad or ProEngineer can easily be converted to .stl format, and the drawing of triangles in OpenGL is simpler than drawing its wireframe model, or even showing its picture.

Chapter 5

MOTION CONTROL

5.1 Introduction

There are six motors of the robot, as mentioned before. These motors are AC servomotors and they are large motors. To drive these motors, there is a electric box with transformer, servo drives and input to this electrical system is 380V. There is a motion control card between this electric box and the computer. This card is Delta-Tau motion control card.

This card is Programmable Multi Axis Controller 2, (PMAC2). This is the high performance servo motion controller, capable of commanding up to eight axis of motion simultaneously with a high level of sophistication. Pmac2 may also run as standalone controller, but in this project, it will be commanded by a computer.

The eight axes can be all associated together for completely coordinated motion or they can be put in its own coordinate system for eight completely independent operations.

There is a user-friendly windows program of the motion control card. From this program, all movements, velocities and accelerations can be followed as well as commanding the motors individually or whole robot at a time. [13]

5.2 Using Motion Control Card

Before using motion control card for running motors, there are some parameters to set for each motor. These parameters are for pre-loading and quality of control of motor. There are proportional, integral and derivative control parameters for its control algorithm. And also there are limits for velocity and acceleration of all the motors individually. After setting all these parameters for the safety and smooth operation, robot is ready for the running. [14]

The parameters were classified in the software for its purpose. I-Variables are initialization and setup parameters, P-Variables are general-purpose user variables, which have global access, Q-Variables are general-purpose user variables that are coordinate specific, and M-Variables are memory access variables.

The user can give line commands for individual motor from the screen, or writes a code in a text file and compile it in the software. Also user can create its own coordinates and commands the motors accordingly.

Every command for each motor is defined previously. Setting I100 variable to 1 means activating the first motor. The velocities and accelerations of the motors can be set from the variable menu or from the program. I116 is the maximum permitted velocity and I117 is the maximum permitted acceleration for the first motor.

Writing motion programs are much easier than writing a computer code in any of the computer languages. There is a motion program example below, and explanations of the lines are at the right sides of the lines:

```
OPEN PROG 1          ; Open buffer for program entry, Program #1
CLEAR                ; Erase existing contents of buffer
LINEAR               ; Blended linear interpolation move mode
ABS                  ; Absolute mode - moves specified by position
TA500                ; Set 1/2 sec (500 msec) acceleration time
TS0                  ; Set no S-curve acceleration time
F5000                ; Set feedrate (speed) of 5000 units(cts) / sec
X10000               ; Move X-axis to position 10000
DWELL500             ; Stay in position for 1/2 sec (500 msec)
X0                   ; Move X-axis to position 0
CLOSE                ; Close buffer - end of program
```

Figure 5.1 Example program for motor control

The position values are the values taken from the encoders of the motors. Approximately 1000 count is equal to 1 degree for METUROBOT’s motors.

5.3 Accessing Motion Control Card from Computer Program

To access Delta Tau motion control card from high-level computer language, i.e. Visual Basic or C++, there is a DLL (Dynamic Link Library) file. A dll is a library of functions, data and resources whose references are resolved at run time of the program. Pmac.dll library contains about 800 functions of the control card. These functions include opening and closing channel for the devise, driving motor, downloading motion program or changing variables from the computer code. These functions can be extracted from the *Pmac.dll* with creating *Pmac.def* (definition) file of the dll. First portion of this Pmac.def file is shown below. [12][18]

```
LIBRARY     PMAC.DLL

EXPORTS
    AddErrRecord                  @1
    AddErrRecordEx                @2
    AutoSetToolOffset             @3
    AutoSetWorkOffset             @4
    AxisToSpindle                 @5
    BackupLinkList                @6
    CalcCoordSys                  @7
    CalculateStepStatistics       @8
    CaptureErrors                 @9
    ClearErrLogFile               @10
...
    OpenPmacDevice                @248
    OpenTextFile                  @249
...
```

Figure 5.2 Some portion of Pmac.def definition file

Before using these functions, they must be imported from the dll library. To do this in c++ code, these lines must be added to the start of the program. In this example, dll file is “Pmac.dll”, function is “OpenPmacDevice(dwnum)”, “dwnum” is device number and first device number is “0”. Note that, Pmac.dll must be located in the system directory.

```
HINSTANCE hinstDLL = LoadLibrary ("PMAC.DLL");
void (FAR *lpfnOpenPmacDevice)(DWORD dwDevice);
FARPROC)lpfnOpenPmacDevice=GetProcAddress(hinstDLL,"OpenPmacDevice
");
```

Figure 5.3 Example code for extracting functions from dll

After writing these lines at the beginning, the function, now, can be used in the code:

```
LpfnOpenPmacDevice(0);
```

Figure 5.4 Example function call

This code opens a channel to a Pmac motion control card.

5.4 Implementation

In the thesis, there are 2 types of motion commands, “point to point” type and “generated path” type. For point to point type, one line command is sufficient. Therefore, the function *“SendLine”* is used for these. In path type movements, a command listing is prepared as a text file and sent to the card at once.

In all these commands, the angular positions of joints are used. The angular position is expressed as the counts of encoders on the motors. For example, for motor #1, a full rotation is about 360.000 counts. That is, if the required rotation of motor #1 is 7 degrees, 7000 cts is used in the code.

In the point to point type movements, the time of movement is not important and starting point as well. The only aim is moving its current position to the end point. For example when the user presses home button, the compiler sends a command line “cc” to the control card.

The line in the code

```
cc=String("#1j=0 #2j=0 #3j=0 #4j=0 #5j=0 #6j=0");
SendLineToRobot(cc);
```

Figure 5.5 Example function call taken from the software (Sending robot to home position)

and, the function of sending line to the card

```
void SendLineToRobot(String strng)
{
 void (FAR *fPmacSendLineA)(DWORD dwDevice,String command);
 (FARPROC)fPmacSendLineA=GetProcAddress(hinstDLL,"PmacSendLineA);
 fPmacSendLineA(0,strng);
};
```

Figure 5.6 Example Function (Sending robot to position using function from dll)

5.6 Conclusion & Future Work

Motion control is sensitive part of this study. After position analyses are done theoretically, the robot must be controlled accurately. Not only the movement, but also the synchronisation is important.

The control card used is Delta Tau multi-axis controller. This control card is very powerful in controlling motors. One card can control up to 8 axis, and 16 card can be used simultaneously with one PC.

The position control is done in the motors. There are very accurate encoders in the motors, and control system of the card is very reliable. The position errors of the motors are about 4-5 counts, it means below 1 percent of the degree.

There are two main drawbacks in the motion control. First of them is the problem with the buffer. When the software of motion control card, “Pewin32”, is not running, the MMI software is getting stuck in some time later. This drawback is about the memory of motion control card. To avoid this problem, MMI software is executed after pewin32.

Other drawback is about getting responses. The commands are sent to the robot very easily, as mentioned before, but MMI cannot get response from the pewin32. This drawback is not creating a big problem right now, because the control system works perfectly. But this can create problems when the robot will be programmed on-line.

Chapter 6

Computer Program

6.1 Introduction

In this chapter, the design, implementation and use of the computer program developed, is described. The MMI (Man Machine Interface) is the most important part of this thesis, because, user controls the robot with this software. The software handles the visual simulation and control of the robot and allows the user to control the move of the robot, point to point or path based. [19]

The software has user interface and some menu items. More important parts of the software, i.e. the items must be controlled by the user are on the screen, and other items, such as joint limits or velocity graphics are under menu items.

There are 4 main parts of the software, robot control, free robot movement control, path generation & execution and simulation panel.

6.2 MMI Software

The main function of the software is to control the robot. Robot can be controlled by giving directly the joint angles or giving the pose of the end-effector. Controlling robot with directly joint angles is the first mission of the software (Forward Control). In this type of control, robot can be moved on-line.

As user changes joint angles, the robot in the screen moves. If user wants to move the real robot, he can choose “move with robot” option, and moves the real robot with the simulation. Also the software saves the position of the robot. then, when the robot is not at the home position at the beginning, the software knows the real position of the robot.

The second thing user can do is controlling the robot by giving position & orientation of the tip point of the robot (Inverse Control). But inverse control is not fast as forward control. The inverse position analysis is done with iterative method, as explained in chapter 2, and it takes some time. Because of this delay, inverse movement cannot be done on-line, but it can be done point by point.

The third thing can be done is path generation. User gives points in space coordinates, with the times of passing, and software generates a path for these points. User can define more than one path, one after the other and also define the waiting time between these paths. This property can be used as used in production lines. After path generation, user can see the position, velocity and acceleration graphics of each joint and tip point. Also, the simulation of robot movement can be seen on the screen and after all steps, movement sent to robot.

There are auxiliary parts of the software, such as robot control and vision properties parts. In robot control section, opening, homing and closing robot buttons exist. User can see the robots status from this part of the screen. The rotation, pan and zooming options are existing for the simulation panel, below the simulation. The last part of the screen is reserved for the information of the pose of the tip point. With the movement of the simulated robot, the values for pose of the tip point changes, for information.

6.3 Interface of the Software

The screenshot of the software is in figure 6.1. There are 5 sections on the main form and there are 5 different menu items.

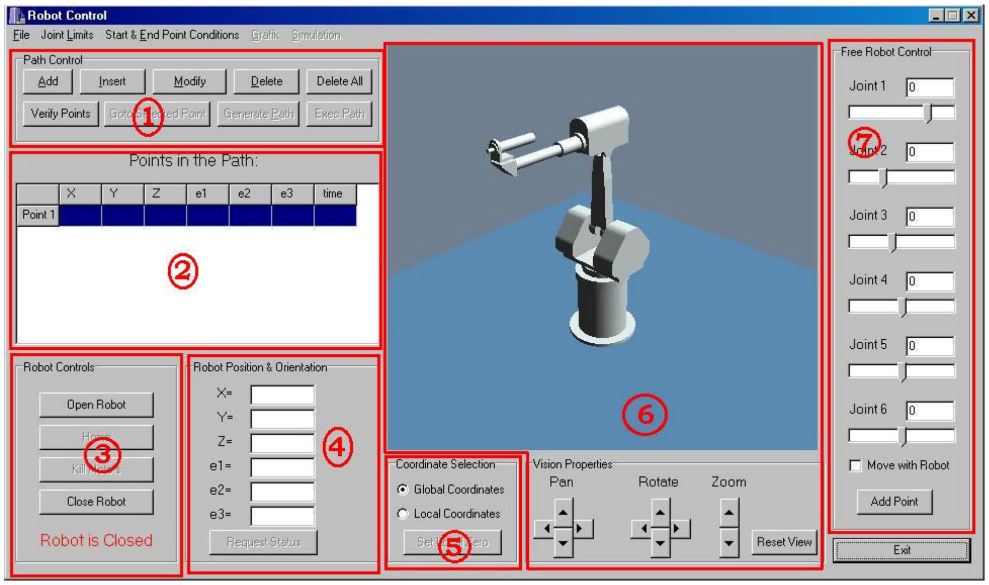


Figure 6.1 Screen view of main form

6.3.1 Controls on the Main Screen

Among controls of the MMI, some of them are on the main form. These are the controls, which have higher priority. The main screen of the software can be divided into 5 parts:

1-2) Point Control Sections: In this part of the software, points in the path are defined. Points are defined in section 1 and shown in section 2. Points are defined in space or coordinates, in either global or local coordinate systems. But points are shown in global coordinate system. X, Y and Z are the elements of the position vector of the tip point. e1, e2 and e3 are euler angles in 123 sequence. In this part, user can add, insert, modify or delete one or more points to the point database. Points in the table can be imported from data file, or exported to the data file.

After filling the table, user must verify all points, i.e. workspace check. If all points are verified, user can see the robot movement on the screen by double-clicking to the relevant line, or send to the real robot by pressing “Goto Selected Point” button.

Path generation is also done in this section. By pressing “Generate Path” button, user generates a path passing these points. If there is no problem, the “Execute Path” button is enabled and by pressing it, user can activate robot.

3) Robot Control: The second part of the software is robot control section. There are 4 buttons in this section. “Open Robot” opens the channel and starts the communication with the robot. “Home” button sends the robot to the home position. “Kill Motors” button kills the robots, that is stopping close-loop control. In every step of control, close-loop control is applied, but before switching the power on, the motors must be killed. Because, if the power is switched on in close-loop controlled motors, the robot jolted. the last button is “Close Robot” button. This closes only the communication to the robot but not closes the robot physically.

The status of the robot can be seen from the text under the “Close Robot” button.

4) Pose of the Tip Point: This box is for only information. Here shown the position and orientation of the tip point. This part works with the simulation panel and shows position values in space coordinates. Units of first three row is mm. Orientation of the tip point is shown by euler angles. It is shown in 1-2-3 sequence and unit is in degrees.

By pressing “Request Status” button on this box, the actual position of the robot can be taken.

5) Coordinate Selection: In this section, user can select either global or local coordinate system. Local zero is defined with pressing “Local Coordinates” radio button. User can also defines its own zero by pressing “Set Local Zero” button.

6) Simulation Panel: Here shows the 3D model of the robot. The movements of the robot can be seen from this window. This window is updated either forward or inverse control of the robot. In forward control, it is updated automatically, in inverse control, after the points are verified, double-clicking point table will update the simulation. When the path is generated the menu “Simulate” enabled and by pressing this menu item, path is simulated in this window.

There are vision properties at the bottom of the panel. User can change viewing direction and zoom to the robot. The “Reset View” button resets the viewing parameters if it is needed.

7) Free Robot Control: In this section, forward robot control is processed. With the help of the slide bars, user can control each joint separately. Changing any joint variable updates the simulation panel and pose of the tip point section. If the robot is open and the check “Move with Robot” is checked, real robot can be controlled on-line.

With the “Add Point” button, the position of the robot can be added to the point list on the second box.

6.3.2 Menu Items

The controls which are not very important, constants or variables like joint limits are buried into the menu items. There are 5 headings in the menu. It is shown in figure 6.2.

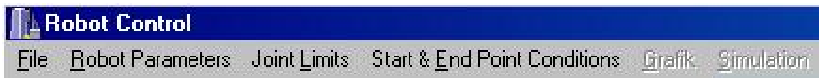
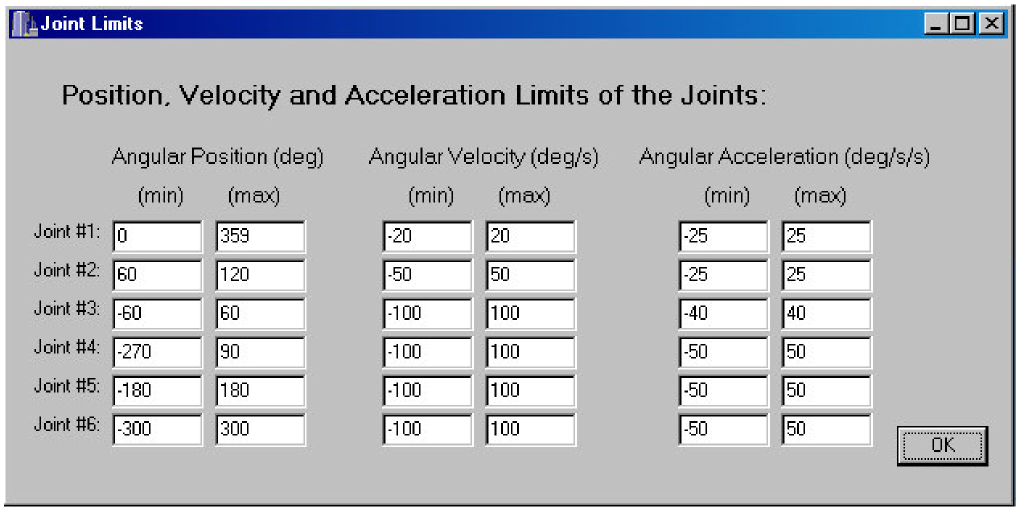


Figure 6.2 Menu Items

1) File: In this menu, open data file, close data file and exit is exist. Data files are points and these are in txt format.

2) Robot Parameters: Robot link-lengths and configuration can be controlled from this tab. Also the length of end effector can be adjusted from the robot link-length section.

3) Joint Limits: When this menu is pressed, the window came on to the screen, involving position, velocity and acceleration limits of the joints. An example of the window is below, figure 6.3. User can change whichever he wants. Position limits are used in inverse kinematics analyses, all are used in path generation as well. When joint limits are updated, points must be verified and path must be generated from the beginning.

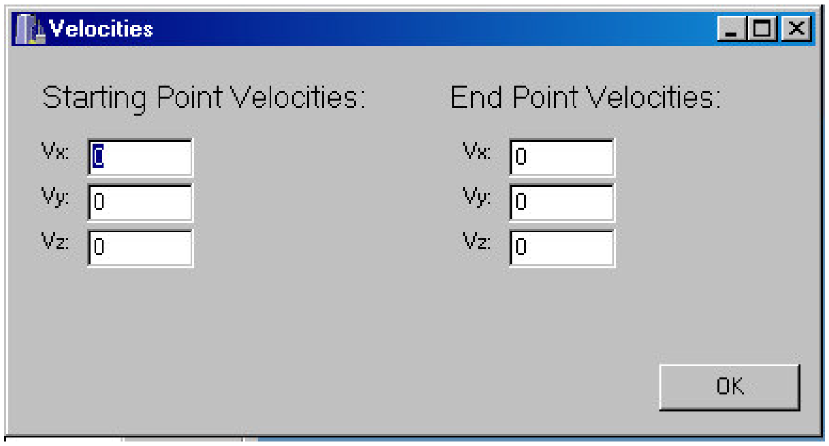

Joint Limits

Position, Velocity and Acceleration Limits of the Joints:

	Angular Position (deg)		Angular Velocity (deg/s)		Angular Acceleration (deg/s/s)	
	(min)	(max)	(min)	(max)	(min)	(max)
Joint #1:	0	359	-20	20	-25	25
Joint #2:	60	120	-50	50	-25	25
Joint #3:	-60	60	-100	100	-40	40
Joint #4:	-270	90	-100	100	-50	50
Joint #5:	-180	180	-100	100	-50	50
Joint #6:	-300	300	-100	100	-50	50

Figure 6.3 Joint Limits Window

4) Start & End Point Conditions: This tab is for path generation only. If start & end point is not stationary, but they have velocities or accelerations or both, these velocities and accelerations are defined here. End point with acceleration has no sense in robotics, but velocity in end points can be used in conveyor systems. The example window is below, figure 6.4.


Velocities

Starting Point Velocities:		End Point Velocities:	
Vx:	0	Vx:	0
Vy:	0	Vy:	0
Vz:	0	Vz:	0

Figure 6.4 Start & End Point Conditions window

5) Graphics: This tab is enabled when the path is generated. There exist position, velocity and acceleration graphic of all joints are exist. These are all in joint coordinates. The values of positions, velocities and accelerations of the joints can be seen with the limits in the same window. As an example, graph of position of joint #3 with the limits is shown in the window below, figure 6.5.

Also the position & velocity graphics of tip point in space coordinates are also exist.

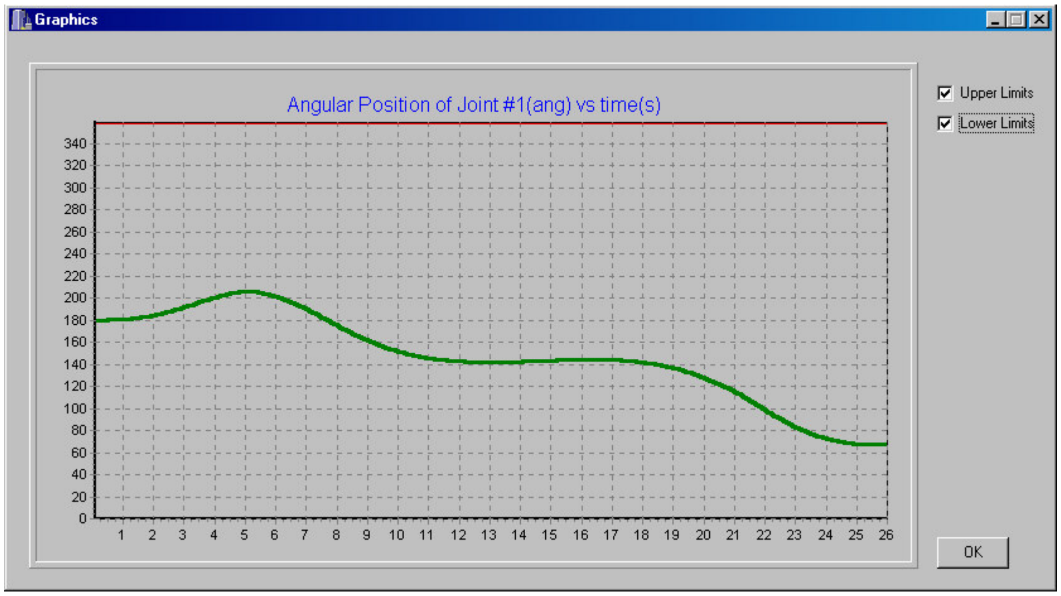


Figure 6.5 Example window of Graphics tab.

6) Simulation: This tab is enabled when the path is generated and joint limits are checked. This is the last step before sending path to the robot. When this button is pressed, the simulation of path starts in the simulation panel. If there is no problem in simulation on the screen, path can be sent to the real robot with the help of appropriate button.

6.4 Subroutines used in Software

There are number of subroutines in the software. Some of them are used alone directly and some of them are used in cooperated. These are classified as functions for kinematic analyses, functions for trajectory planning, functions for controlling robot and functions for OpenGL simulation. There are brief explanations of the functions below. The usages of the functions are in the appendix.

There are number of functions used for kinematic analyses. Inverse position analysis is done in the function "*Inverse*". Forward position analysis is done in "*Draw*" function. While drawing to the screen, the position and orientation is calculated. There are "*Jacobian*" and "*dJacobian*" functions to calculate the jacobian matrices and its derivative.

For the robot control, the main function is "*SendLineToRobot*" function. It sends a command line to the robot. "*OpenPmacDevice*", "*ClosePmacDevice*" and "*DownloadFile*" are the other commands used in the software.

The main subroutine used for the trajectory is under the button "*Generate Path*". There is no independent function, but there are auxiliary functions for this code. For matrix operation, "*Jacobian*" and "*dJacobian*" functions creates the Jacobian matrices and "*Mat_Inv*" inverses 6x6 matrix. "*Mult_Mat*" is the function for multiplying matrices. These functions are designed for only the trajectory planning, except Mat_Inv. This is the function for inversion of all matrices. After path is generated, limits are checked with "*Check_Limits_X*" function. X denotes the number of degree of the polynomial fitted, either 3 or 4.

The only function user must know about is "*Draw*" function for visual simulation. But there are many functions used for initializing, importing and drawing robot. "*InitWindow*" is used for initialization. "*Begin_Draw*", "*End_Draw*" and "*Draw_Robot*" are used for drawing robot into the window.

“*Draw_Link*” subroutine import links of the robot. There are 3 more functions for the drawing of robot. “*VmV*”, “*VcrossV*” and “*Vnormalize*”. These are auxiliary functions for OpenGL. Some of the functions for visual simulation are taken from the thesis of Anas Abidi [1].

6.5 User Guide of Software

Using MMI of METUROBOT is not so complicated, therefore there is no need to explain it in independent chapter. The main idea of software is controlling the robot. After opening communications with robot, user can control the robot in 3 ways. *Free robot control*, *point to point control* and *path control*.

The interface of the software is shown below (Figure 6.6):

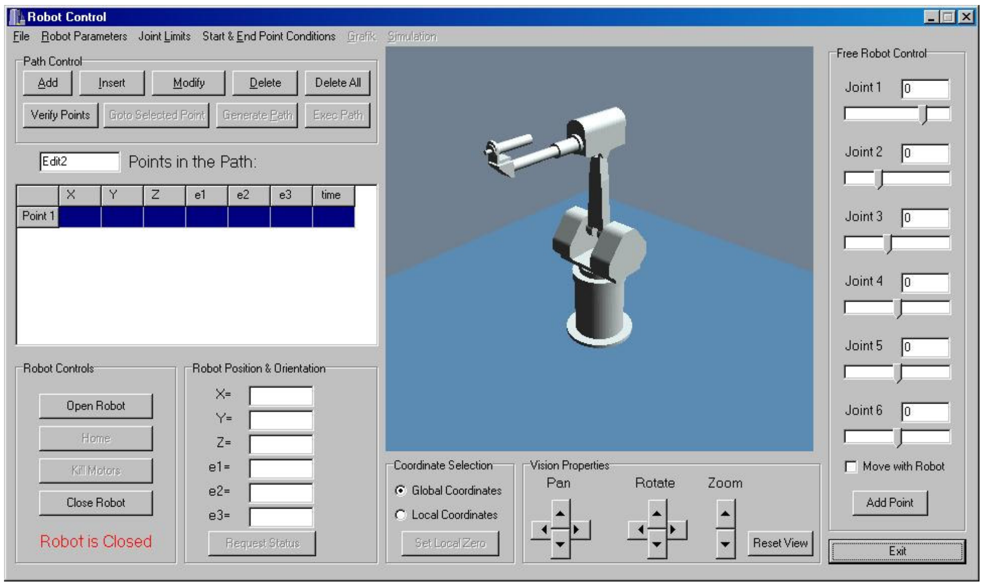


Figure 6.6 Interface of software.

- **Opening Robot:** Before giving power to the control unit of the robot, software must be executed. Then, by pressing “*Open Robot*” button on the robot control tab, communications are opened. Then with the "*Kill Motors*" button, the motors must be released. After these, power can be supplied to the control unit of the robot.

- **Free Move:** For the free movement of the robot, the sliders on the right hand side of the screen are used. In this type of control, each joint is controlled separately. If the checkbox "*Move with Robot*" is checked, real robot will move simultaneously with the simulation on screen. In any type of movement, the position & orientation of the tip point is updated.

- **Point to Point Control:** Before sending robot to any position, points must be defined and verified. There is a "*Points Table*" on the screen. User can import points from the file, or inputs point by point from the buttons above the table. After all the points are written on the table, points must be verified with the button "*Verify Points*". If all the points are verified, "*Go to Selected Point*" button is enabled. Pressing this button commands robot to go to this point. If user wants to check the position of the point before execution, he can see the position of the robot of the screen by double-clicking the point on the table.

- **Path Control:** If the user wants to execute the robot on the path, first, points on the path must be entered to the table. After verifying points, user can generate path with the button "*Generate Path*". For this step, the times of pass must be entered on the table. If the path is generated without any problem, the tabs "*Graphics*" and "*Simulation*" are enabled. In the graphics tab, the user can examine the position, velocity and accelerations of the each joint. With the simulation tab, user can see the planned path on the screen. If the user verifies the path, “*Execute Path*” button sends this path to the robot, and robots starts to move along this path.

- **Request Status:** To inquire the robots actual position, the “Request Status” button is used. This button can be used anytime, when the robot is open.

- **Closing Robot:** After all processes, the robot must be closed. Before unplugging the robot. The robot must be sent to home-position by pressing the “*Home*” button on the screen and close the robot with “*Close Robot*” button. Then it is safe to unplug the Robot.

6.6 Some Remarks About the Software

There are some “rules” to obey, when the program is running. These rules must be obeyed in case of any bad luck of the robot. Some of the rules can be eliminated by developing software further. The “rules” of the robot to be paid attention are in below.

- Run “PeWin32” software before executing the MMI software,
- Kill motors before switching the robot on,
- Check the position of the point, or generated path before send command to the robot.
- Do not give fast commands while running in “Free Control” mode.
- “Home” the robot before shutting down the computer for the safety.

6.7 Conclusion

In this chapter, the MMI software is explained. The subject of the thesis is mainly developing this software. The software developed is user-friendly and windows based. The simulation on the screen is successful and the software works without any errors. There are some incomplete parts, but in general, the software is sufficient for the main purposes. The suggestions for the additions to the software are discussed in the last chapter, but with the existing form of the executable file, it avoids the unwanted movements or accidents with the robot.

Chapter 7

Error Analyses

7.1 Introduction

In this chapter, the error sources in the software and robot system will be investigated. The errors in the whole system can be explained in three main categories;

- errors due to the construction of the robot,
- errors due to the method used in inverse position analyses,
- and, errors due to the motion control.

7.2 Errors due to the Construction of the Robot

The robot concerned (METUROBOT) is designed and manufactured in METU CAD/CAM Centre. There can be some production and assembly errors and misalignments. This can cause some error in positioning in global coordinate system. These errors are constant errors but the error in the parts of assembly cannot be measured accurately. Without disassembling the robot. These are not creating big errors, but this is the part of the error of the system.

The foundation of the robot has some tilt. This causes also some positioning errors. The links are bending a little under loading conditions and under its own weight, when there is a tilt. This is not a constant error, because, when there are more loads or when the robot is trying to reach the position far from home position, the error increases.

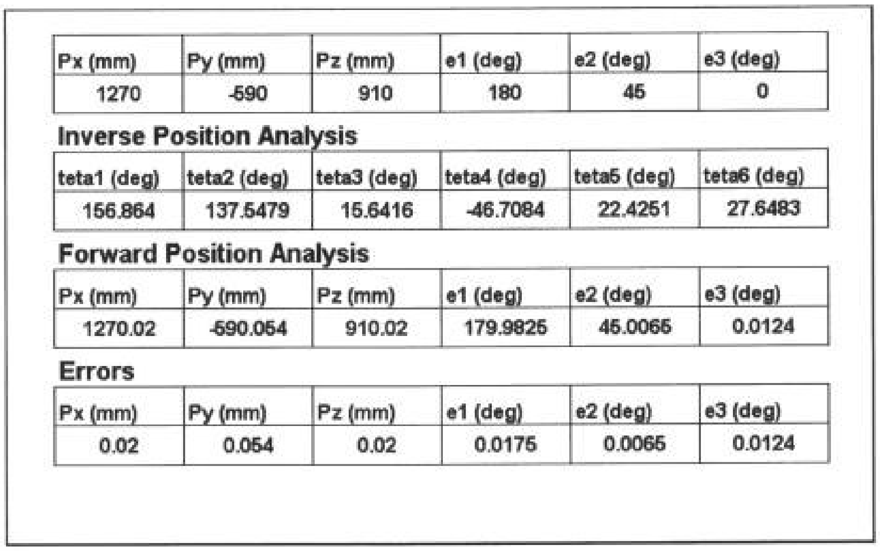
7.3 Errors due to the Inverse Position Analyses

Some error occurs in the inverse position analysis. This error is due to the search algorithm. The angles, θ_2 to θ_6 are determined in terms of θ_1. Then θ_1 is found by search algorithm. To determine true θ_1 we must search in the range 0 to 360 degrees. The inverse interpolation method is used in this search algorithm. This method is very accurate, but it causes a little error.

The second error is mathematical error. In the algorithm, there are lots of calculations, squares, square roots, sine, cosine and atan2 functions. These functions all generate some calculation errors.

Below, there are some outputs of inverse position analysis (Table 7.1&7.2). First step of the test is inverse position analysis and then we make forward position analysis to verify this points. The error is calculated in mm for positions and degrees for orientation.

Table 7.1 Example 1 for Inverse Position Analysis



Px (mm)	Py (mm)	Pz (mm)	e1 (deg)	e2 (deg)	e3 (deg)
1270	-690	910	180	45	0

Inverse Position Analysis

teta1 (deg)	teta2 (deg)	teta3 (deg)	teta4 (deg)	teta5 (deg)	teta6 (deg)
156.864	137.5479	15.6416	-46.7084	22.4251	27.6483

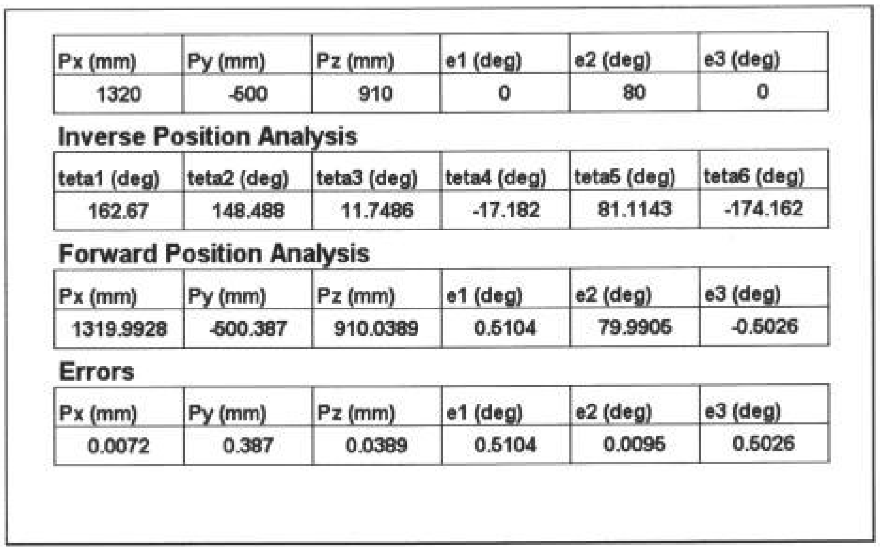
Forward Position Analysis

Px (mm)	Py (mm)	Pz (mm)	e1 (deg)	e2 (deg)	e3 (deg)
1270.02	-690.054	910.02	179.9825	45.0065	0.0124

Errors

Px (mm)	Py (mm)	Pz (mm)	e1 (deg)	e2 (deg)	e3 (deg)
0.02	0.054	0.02	0.0175	0.0065	0.0124

Table 7.2 Example 2 for Inverse Position Analysis



Px (mm)	Py (mm)	Pz (mm)	e1 (deg)	e2 (deg)	e3 (deg)
1320	-500	910	0	80	0

Inverse Position Analysis

teta1 (deg)	teta2 (deg)	teta3 (deg)	teta4 (deg)	teta5 (deg)	teta6 (deg)
162.67	148.488	11.7486	-17.182	81.1143	-174.162

Forward Position Analysis

Px (mm)	Py (mm)	Pz (mm)	e1 (deg)	e2 (deg)	e3 (deg)
1319.9928	-500.387	910.0389	0.5104	79.9905	-0.5026

Errors

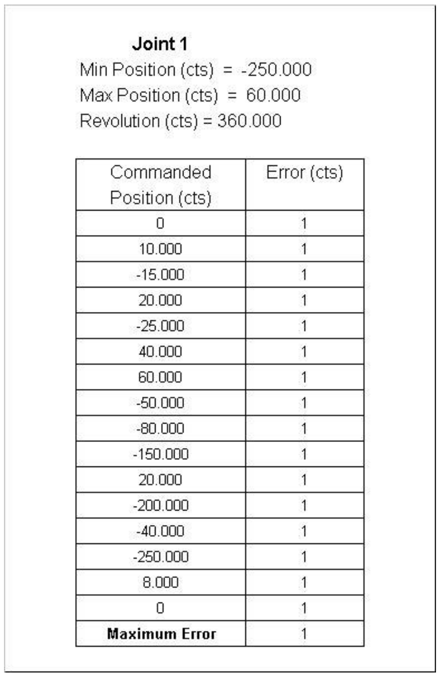
Px (mm)	Py (mm)	Pz (mm)	e1 (deg)	e2 (deg)	e3 (deg)
0.0072	0.387	0.0389	0.5104	0.0095	0.5026

7.4 Errors due to the Motion Control

There are some errors on the motion control, i.e. commanded and actual positions are not the same. But the control system of Delta-Tau motion control card is powerful and the error due to motor control is very little.

To determine the value of motor control, an experiment is done. Motors are commanded to some positions in their range, and error is examined on the counter indicators of the Pewin32 software. The results of the experiment are in below. Each box represents each motor. (Tables 7.3-7.8). In each box, the position limits of the motors, the revolution distance, commanded positions and examined errors are existing.

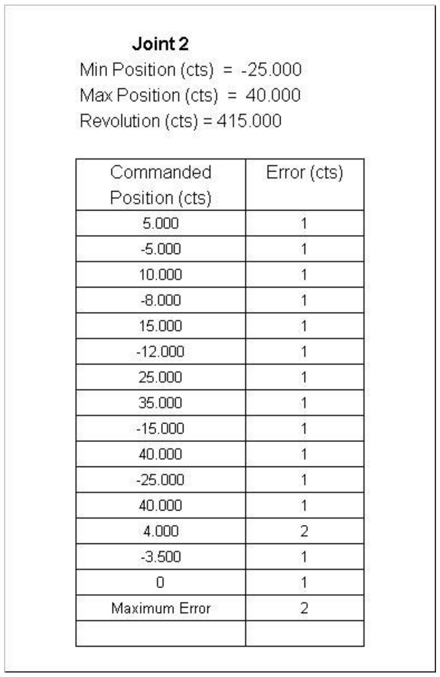
Table 7.3 Error analysis of motor #1



Joint 1
Min Position (cts) = -250.000
Max Position (cts) = 60.000
Revolution (cts) = 360.000

Commanded Position (cts)	Error (cts)
0	1
10.000	1
-15.000	1
20.000	1
-25.000	1
40.000	1
60.000	1
-50.000	1
-80.000	1
-150.000	1
20.000	1
-200.000	1
-40.000	1
-250.000	1
8.000	1
0	1
Maximum Error	1

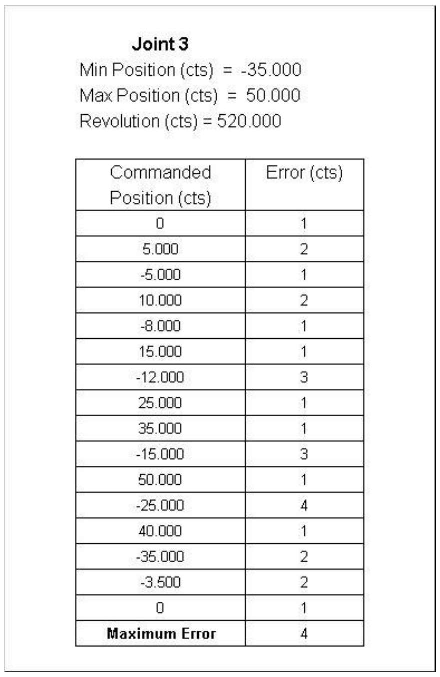
Table 7.4 Error analysis of motor #2



Joint 2
Min Position (cts) = -25.000
Max Position (cts) = 40.000
Revolution (cts) = 415.000

Commanded Position (cts)	Error (cts)
5.000	1
-5.000	1
10.000	1
-8.000	1
15.000	1
-12.000	1
25.000	1
35.000	1
-15.000	1
40.000	1
-25.000	1
40.000	1
4.000	2
-3.500	1
0	1
Maximum Error	2

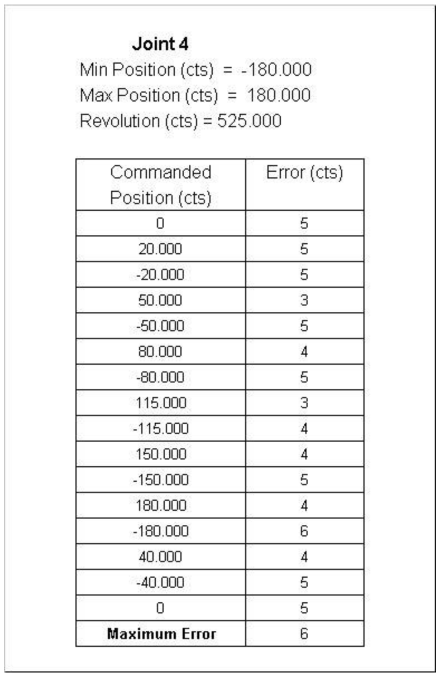
Table 7.5 Error analysis of motor #3



Joint 3
Min Position (cts) = -35.000
Max Position (cts) = 50.000
Revolution (cts) = 520.000

Commanded Position (cts)	Error (cts)
0	1
5.000	2
-5.000	1
10.000	2
-8.000	1
15.000	1
-12.000	3
25.000	1
35.000	1
-15.000	3
50.000	1
-25.000	4
40.000	1
-35.000	2
-3.500	2
0	1
Maximum Error	4

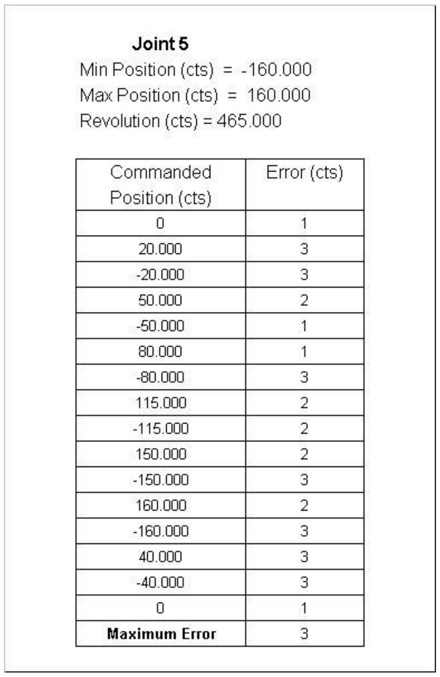
Table 7.6 Error analysis of motor #4



Joint 4
Min Position (cts) = -180.000
Max Position (cts) = 180.000
Revolution (cts) = 525.000

Commanded Position (cts)	Error (cts)
0	5
20.000	5
-20.000	5
50.000	3
-50.000	5
80.000	4
-80.000	5
115.000	3
-115.000	4
150.000	4
-150.000	5
180.000	4
-180.000	6
40.000	4
-40.000	5
0	5
Maximum Error	6

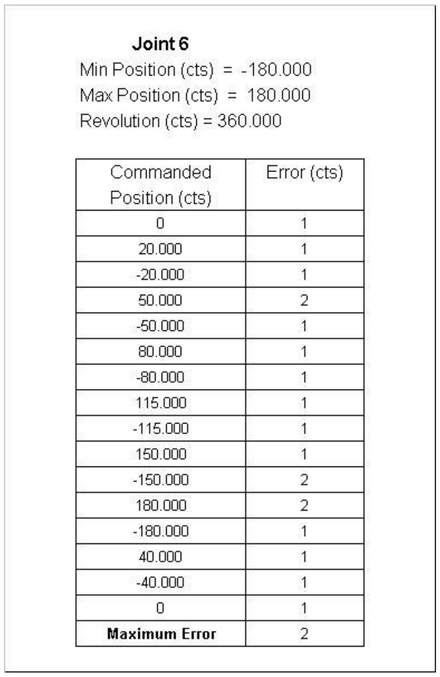
Table 7.7 Error analysis of motor #5



Joint 5
Min Position (cts) = -160.000
Max Position (cts) = 160.000
Revolution (cts) = 465.000

Commanded Position (cts)	Error (cts)
0	1
20.000	3
-20.000	3
50.000	2
-50.000	1
80.000	1
-80.000	3
115.000	2
-115.000	2
150.000	2
-150.000	3
160.000	2
-160.000	3
40.000	3
-40.000	3
0	1
Maximum Error	3

Table 7.8 Error analysis of motor #6



Joint 6
Min Position (cts) = -180.000
Max Position (cts) = 180.000
Revolution (cts) = 360.000

Commanded Position (cts)	Error (cts)
0	1
20.000	1
-20.000	1
50.000	2
-50.000	1
80.000	1
-80.000	1
115.000	1
-115.000	1
150.000	1
-150.000	2
180.000	2
-180.000	1
40.000	1
-40.000	1
0	1
Maximum Error	2

According to these results, errors are very little. The maximum error on the experiments are 6 counts, this is about 4/1000 of a degree. These errors may be made even smaller by improving the control system of motion control card.

7.5 Error Analysis, Accuracy and Repeatability of the METUROBOT

Accuracy and repeatability of the robot is in terms of micrometers, because the encoders in the motors are highly sensitive. But there are error sources in the system. Error of the robot is determined with the measurement. For this purpose, the steel ruler is used. The steel string is attached to the tip point of the robot as a pointer and the position of the tip point is commanded to one position to the other. The setup is shown in figure 7.1, below.

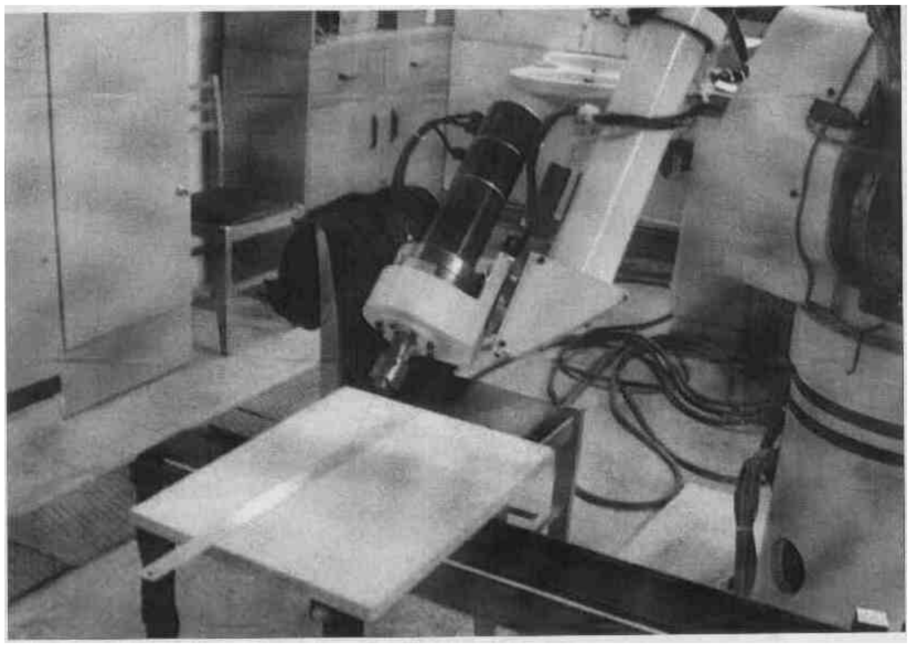


Figure 7.1 Setup for the measurement of error.

The thickness of the steel string is 1 mm, and the scale of the steel ruler is 1 mm, therefore we can observe the errors in mm.

In the experiment, the tip point of the robot is brought to the starting point, which is 200-mm. line in the ruler. Then the command is to go 40 mm from this point in the direction of the ruler. The observed error is below 1 mm. Two positions of the robot are shown in the following figures, figure 7.2 and 7.3. As a result, the error in the system is much higher than the accuracy and repeatability.

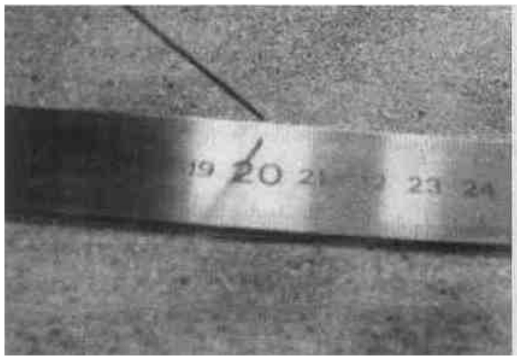


Figure 7.2 Start of the experiment

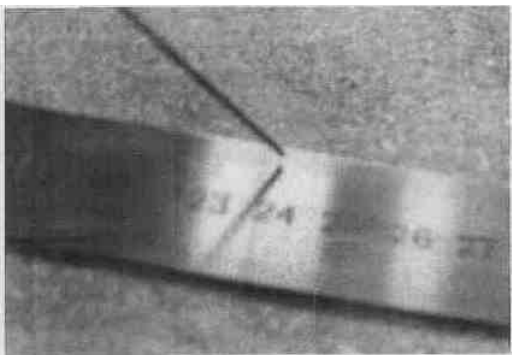


Figure 7.3 End of the experiment

7.6 Conclusion

The error comes out in all robotic systems. In fact, there are errors in all mechanical systems. These errors are generally due to the production and wear. The METUROBOT has also some errors due to production, assembly and wear. User cannot use METUROBOT as a high precision machine, but the error is not big, so the robot can be used in pick and place works.

The major part of the error is constant. Therefore, repeatability of the robot is very good. If these errors can be avoided, METUROBOT can be used as a high precision robot.

Chapter 8

Conclusion

8.1 Discussion & Conclusion

In this thesis project, Man-Machine-Interface of an industrial robot is developed. The thesis has mainly three parts, the kinematics theory of the robot, motion control and software. The robot considered is designed and manufactured in METU. For this reason, kinematic analyses are done first time. In addition to this, inverse position analysis cannot be solved by fully analytical methods, and an iterative method, semi-analytical method had to be used. This part can be reconsidered later. Also path generation is examined and applied in the thesis, but path optimisation is not considered.

Motion control of the robot is discussed in chapter 5. The control system of robot is very powerful, but control from the C++ compiler is not that much powerful. Using set of these cards, one can control up to 128 motors, but card is rather old. Windows95/98 cannot recognise the card and the risk of coincide with other hardware exists. There are more than 800 functions available to use, but in this thesis, only few of them are used. For other applications, more of them can be used.

Details of the software are explained in chapter 6. It is easy to use and its interface is explanatory. But the code has no certain start or end point, because the code is object-oriented. Therefore, user must know robotics to use the program, especially controlling real robot parts. The C++ is object oriented and functions are used in the code. The code can be developed further easily with this construction.

8.2 Future Work

In this part, some suggestions for the future work on the theory & software is listed.

- In the thesis, path optimisation is not considered. It can be considered as a future work of the system.
- Energy optimisation can be applied to the path planning.
- Collusion detection is not considered. Collusion detection and avoidance can be applied to the thesis.

REFERENCES

1. ABIDI Anas, “Man-Machine Interface Software Development for an Industrial Robot”, METU Mechanical Engineering Master of Science Thesis, 2002.

2. Balkan T., Özgören M. K., Arıkan M. A. S., Baykurt H. M., “A Method of Inverse Kinematics Solution Including Singular and Multiple Configurations for a Class of Robotic Manipulators”, Mechanism and Machine Theory, pp.1221-1237, 2000.

3. Çağlayan M. Erdal, “Computer Aided Design of an Industrial Robot Arm”, METU Mechanical Engineering Master of Science Thesis, 1994.

4. Eren Oykun, “Production, Assembly and Application of an Industrial Robot”, METU Mechanical Engineering Master of Science Thesis, 2001.

5. Kelly Derek, “A Layman’s Introduction To Robotics”, Petrocelli Books, 1986.

6. Konukseven E. İlhan, “Graphical Simulation and Programming of Robots”, METU Mechanical Engineering Master of Science Thesis, 1989.

7. Koren Yoram, “Robotics For Engineers”, McGraw-Hill Book Company, 1985.

8. Neider J., Davis T., Woo M., "OpenGL Programming Guide", Addison-Wesley Publishing Company, 1994.

9. OpenGL Panel Download & Examples, http://www.allanpetersen.com/opengl.htm

10. Özgören M. Kemal, "ME 522 Lecture Notes on Principle of Robotics", METU, unpublished 2001.

11. Özgören M. Kemal, "Topological Analysis of 6-joint Serial Manipulators and Their Inverse Kinematic Solutions", Mechanism and Machine Theory, vol.37, No.5, ppç511-548, 2002.

12. "Pcomm32, PMAC 32 Bit Driver" manual, Delta Tau Systems, 2000

13. "Pewin32, PMAC Executive for Windows" manual, Delta Tau Stayems, 1995.

14. "PMAC Users Manual", Delta Tau Systems Inc, 1991.

15. Schildt, H., "Borland C++ : the complete reference", McGraw Hill, 1997.

16. Toker Ş. Bülent, "Virtual Modelling, Planning and Production of Parts of an Industrial Robot", METU Mechanical Engineering Master of Science Thesis, 1999.

17. Ünver Tolga, "The Computer Aided Design of an Industrial Robot", METU Mechanical Engineering Master of Science Thesis, 1997.

18. Valley, S., “ObjectWindows : Programming Guide”, Borland Inrenational, 1992.

19. Zomaya A.Y., “Modelling and Simulation of Robot Manipulators”, World Scientific, 1992.

Appendix A

Functions Developed and Used in the Software

A.1 Introduction

Here are the functions used in the software. These can be classified as robot control functions, kinematic analysis functions, path generation functions and functions for simulation framework. The function can return one value at a time, therefore, some functions return nothing, but stores result in the global variables.

A.2 Robot Control Functions

These functions are for the robot control. Robot is controlled by controlling each motor separately. These functions are global functions and they can be used in any code for this robot. But when using in the other software, functions must be imported from the Pmac.dll file. This procedure is explained in chapter 5.3.

- ***fOpenPmacDevice(int dw);*** is the function for opening robot & initialisation. It is extracted from Pmac.dll file and used directly in the program. dw is the device number, and it is always 0 for this thesis.

- ***fClosePmacDevice(int dw);*** is the function for closing communication with the robot. This function is also directly called from the Pmac.dll file.

- ***fPmacDownloadFile(int dw, string cc);*** is the function for downloading file to the motion control card. This function is used for downloading gains before controlling motors and downloading motion programs for path execution. cc is the path of the file to be downloaded.

A.3 Kinematic Analysis Functions

There are kinematic analysis functions in this part. These functions are the main parts of this study. The inverse position analysis function can be improved later.

- ***Inverse(double Px, double Py, double Pz, double e1, double e2, double e3);*** is the function for inverse position analysis. Px, Py & Pz are the tip point positions and e1, e2 & e3 are the euler angles for the tip point orientation. The sequence is 1-2-3 sequence. When this function called, the resultant joint angles are stored in *t1_f*, *t2_f*, *t3_f*, *t4_f*, *t5_f* & *t6_f* variables. Px, Py & Pz are in mm, and e1, e2 & e3 are in degrees.

- ***Draw(double ang1, double ang2, double ang3, double ang4, double ang5, double ang6);*** is the function for forward position analyses & drawing robot to the screen. *angX* variables are the joint variables in *radians*.

- ***Jacobian(double t1, double t2, double t3, double t4, double t5, double t6);*** is the function for finding jacobian matrix. tX's arejoint variables in radians and jacobian matrix is stored in matrix variable Jacob[][].

- ***dJacobian(double t1, double dt1, double t2, double dt2, double t3, double dt3, double t4, double dt4, double t5, double dt5, double t6, double dt6);*** is the function for finding derivative of jacobian matrix. tX's are joint varianles in radians and dtX's are derivative of joint variables, i.e. velocities of joints at that moment. Derivative of jacobian matrix is stored at dJacob[][] matrix.

A.4 Path Generation Functions

There is no independent function for path generation. The path generation is under the click event of the "Generate Path" button. In this event, there are some functions, such as matrix inversion, multiplication or joint limit check.

- ***Mat_Inv(int n, double A[100][100]);*** is the function for matrix inversion. n is the size of the matrix, and A[][] is the input matrix. The inverse of the matrix is stored at global InMat[][] matrix. This code can be used other than this study.

- ***MultMat(int k,double A_carp[100][100],double B_carp[100];*** is the function for multiplying (k x k) matrix with kx1 matrix. This type of operation is needed for path generation.

- ***Check_Limits_4(int count, double ee, double dd, double cc, double bb, double aa, int timedur);*** is the function for the joint limit check of the generated path. This function is used for 4th order polynomials. The "count" variable is the joint number. "ee" to "aa" are the constants of the 4th order polynomial fitted to the interval and "timedur" is the duration of the interval.

- ***Check_Limits_3(int count, double dd, double cc, double bb, double aa, int timedur);*** is the same function as above. The only difference is that, this function is 3rd order polynomials.

A.5 Functions for Simulation Framework

The main function for the simulation part is the Draw function. It is explained in the part A.3, because it includes forward position analysis also. The only function user must know is this function. But inside the Draw function, there are functions for importing models, drawing to the panel, etc. These functions can be classified as advanced functions. For one, who will improve the simulation framework, there are explanations of these functions.

The functions are interconnected to each other. Therefore, while modifying the simulation framework part of this study, the operator must be very careful. All these functions must be carried together.

- ***InitWindow();*** function initialises the window for OpenGL. It is copied to the *onPaint* event of the Panel. The position of light and base colour of the objects are designated here.

- ***BeginDraw();*** function clears buffers and prepares the panel to the drawing.

- ***EndDraw();*** function finishes the drawing and again clears the temporary used buffers.

- ***VmV(), VcrossV(), Vnormalize();*** are the functions for finding normal of the surfaces. As explained before, all objects are made up of triangles and for all triangles, the colour variation is determined by using this normal of the surface and position of the light determined.

- ***Draw_Link(char *rawfile, int n);*** is the function for importing links. **rawfile* is the path of the .raw file for link & *n* is the link number. This function open the data file, extract the triangles and draws it to the screen.

- ***Rotate_Link(int n)***; is the same as the *Draw_Link*, but it only updates the view for new position.

- ***Draw_Robot();*** is the main function of drawing robot to the panel. This function calls the other functions for drawing robot to the panel.

Appendix B

Jacobian Matrix and its Derivative

B.1 Introduction

For the velocity and acceleration analyses, jacobian matrix and its derivative are used. The jacobian matrix is the matrix in terms of joint angles and its derivative is in terms of joint angles and its derivative, i.e. the angular position and velocities of the joints.

B.2 Jacobian Matrix

The definition of jacobian matrix is;

$$J_P = \begin{pmatrix} J_{P1} & J_{P2} & J_{P3} & J_{P4} & J_{P5} & J_{P6} \\ J_{A1} & J_{A2} & J_{A3} & J_{A4} & J_{A5} & J_{A6} \end{pmatrix}$$

where, $J_{Pn} = \frac{\delta}{\delta q_n} P$

$$J_{An} = \mathrm{col}\left[\left(\frac{\delta}{\delta q_n} C\right) \cdot C^T\right]$$

This was equation (2.56). the J_{Pn} and J_{An} are the sub-matrices of size 1x3. P and C matrices are defined in (eqs 2.11-2.19). When we carry out the calculations, the results are,

$$J_{A1} = \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}$$

$$J_{A2} = \begin{pmatrix} s\theta_1 \\ -c\theta_1 \\ 0 \end{pmatrix}$$

$$J_{A3} = \begin{pmatrix} s\theta_1 \\ -c\theta_1 \\ 0 \end{pmatrix}$$

$$J_{A4} = \begin{pmatrix} c\theta_1 \cdot s\theta_{23} \\ s\theta_1 \cdot s\theta_{23} \\ -c\theta_{23} \end{pmatrix}$$

$$J_{A5} = \begin{pmatrix} c\theta_1 \cdot s\theta_4 \cdot c\theta_{23} - s\theta_1 \cdot c\theta_4 \\ s\theta_1 \cdot s\theta_4 \cdot c\theta_{23} + c\theta_1 \cdot c\theta_4 \\ s\theta_{23} \cdot s\theta_4 \end{pmatrix}$$

$$J_{A6} = \begin{bmatrix} -c\theta_1 \cdot \left(c\theta_4 \cdot s\theta_5 \cdot c\theta_{23} - c\theta_5 \cdot s\theta_{23}\right) - s\theta_1 \cdot s\theta_4 \cdot s\theta_5 \\ -s\theta_1 \cdot \left(c\theta_4 \cdot s\theta_5 \cdot c\theta_{23} - c\theta_5 \cdot s\theta_{23}\right) + c\theta_1 \cdot s\theta_4 \cdot s\theta_5 \\ -c\theta_4 \cdot s\theta_{23} \cdot s\theta_5 - c\theta_{23} \cdot c\theta_5 \end{bmatrix}$$

and,

$$J_{P1} = \begin{bmatrix} -a_2 \cdot s\theta_1 \cdot c\theta_2 - d_3 \cdot s\theta_1 \cdot s\theta_{23} - d_4 \cdot (s\theta_1 \cdot c\theta_{23} \cdot s\theta_4 + c\theta_1 \cdot c\theta_4) + d_p \cdot (s\theta_1 \cdot c\theta_{23} \cdot c\theta_4 \cdot s\theta_5 - c\theta_1 \cdot s\theta_4 \cdot s\theta_5 - s\theta_1 \cdot s\theta_{23} \cdot c\theta_5) \\ a_2 \cdot c\theta_1 \cdot c\theta_2 + d_3 \cdot c\theta_1 \cdot s\theta_{23} + d_4 \cdot (c\theta_1 \cdot c\theta_{23} \cdot s\theta_4 - s\theta_1 \cdot c\theta_4) - d_p \cdot (c\theta_1 \cdot c\theta_{23} \cdot c\theta_4 \cdot s\theta_5 + s\theta_1 \cdot s\theta_4 \cdot s\theta_5 - c\theta_1 \cdot s\theta_{23} \cdot c\theta_5) \\ 0 \end{bmatrix}$$

$$J_{P2} = \begin{bmatrix} -a_2 \cdot c\theta_1 \cdot s\theta_2 + d_3 \cdot c\theta_1 \cdot c\theta_{23} - d_4 \cdot c\theta_1 \cdot s\theta_4 \cdot s\theta_{23} + d_p \cdot c\theta_1 \cdot (c\theta_4 \cdot s\theta_5 \cdot s\theta_{23} + c\theta_5 \cdot c\theta_{23}) \\ -a_2 \cdot s\theta_1 \cdot s\theta_2 + d_3 \cdot s\theta_1 \cdot c\theta_{23} - d_4 \cdot s\theta_1 \cdot s\theta_4 \cdot s\theta_{23} + d_p \cdot s\theta_1 \cdot (c\theta_4 \cdot s\theta_5 \cdot s\theta_{23} + c\theta_5 \cdot \theta_{23}) \\ a_2 \cdot c\theta_2 + d_3 \cdot s\theta_{23} + d_4 \cdot c\theta_{23} \cdot s\theta_4 + d_p \cdot (c\theta_4 \cdot s\theta_5 \cdot c\theta_{23} + c\theta_5 \cdot s\theta_{23}) \end{bmatrix}$$

$$J_{P3} = \begin{bmatrix} d_3 \cdot c\theta_1 \cdot c\theta_{23} - d_4 \cdot s\theta_4 \cdot c\theta_1 \cdot s\theta_{23} + d_p \cdot c\theta_1 \cdot (s\theta_2 \cdot c\theta_3 \cdot c\theta_4 \cdot s\theta_5 + c\theta_5 \cdot c\theta_{23}) \\ d_3 \cdot s\theta_1 \cdot c\theta_{23} - d_4 \cdot s\theta_4 \cdot s\theta_1 \cdot s\theta_{23} + d_p \cdot s\theta_1 \cdot (s\theta_2 \cdot c\theta_3 \cdot c\theta_4 \cdot s\theta_5 + c\theta_5 \cdot c\theta_{23}) \\ d_3 \cdot s\theta_{23} + d_4 \cdot c\theta_{23} \cdot s\theta_4 - d_p \cdot c\theta_5 \cdot (c\theta_2 \cdot c\theta_3 \cdot s\theta_4 - s\theta_{23}) \end{bmatrix}$$

$$J_{P4} = \begin{bmatrix} d_4 \cdot (c\theta_1 \cdot c\theta_{23} \cdot c\theta_4 + s\theta_1 \cdot s\theta_4) + d_p \cdot (c\theta_1 \cdot c\theta_{23} \cdot s\theta_4 \cdot s\theta_5 - s\theta_1 \cdot c\theta_4 \cdot s\theta_5) \\ d_4 \cdot (s\theta_1 \cdot c\theta_{23} \cdot c\theta_4 - c\theta_1 \cdot s\theta_4) + d_p \cdot (s\theta_1 \cdot c\theta_{23} \cdot s\theta_4 \cdot s\theta_5 + c\theta_1 \cdot c\theta_4 \cdot s\theta_5) \\ d_4 \cdot s\theta_{23} \cdot c\theta_4 + d_p \cdot s\theta_{23} \cdot s\theta_4 \cdot s\theta_5 \end{bmatrix}$$

$$J_{P5} = \begin{bmatrix} -d_p \cdot (c\theta_1 \cdot c\theta_{23} \cdot c\theta_4 \cdot c\theta_5 + s\theta_1 \cdot s\theta_4 \cdot c\theta_5 + c\theta_1 \cdot s\theta_{23} \cdot s\theta_5) \\ -d_p \cdot (s\theta_1 \cdot c\theta_{23} \cdot c\theta_4 \cdot c\theta_5 - c\theta_1 \cdot s\theta_4 \cdot c\theta_5 + s\theta_1 \cdot s\theta_{23} \cdot s\theta_5) \\ -d_p \cdot (s\theta_{23} \cdot c\theta_4 \cdot c\theta_5 - c\theta_{23} \cdot s\theta_5 \end{bmatrix}$$

$$J_{P6} = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}$$

B.3 Derivative of Jacobian Matrix

Jacobian matrix is the function of all joint variables.

$$dJ_P = \begin{pmatrix} dJ_{P1} & dJ_{P2} & dJ_{P3} & dJ_{P4} & dJ_{P5} & dJ_{P6} \\ dJ_{A1} & dJ_{A2} & dJ_{A3} & dJ_{A4} & dJ_{A5} & dJ_{A6} \end{pmatrix}$$

The derivative of jacobian matrix is the partial derivative of jacobian matrix for all of the variables. The elements of dJ matrix are very long, but how to derive is easy.

$$dJ_{xx} = \left(\frac{d}{d\theta_1}J_{xx}\right)\cdot\dot{\theta}_1 + \left(\frac{d}{d\theta_2}J_{xx}\right)\cdot\dot{\theta}_2 + \left(\frac{d}{d\theta_3}J_{xx}\right)\cdot\dot{\theta}_3 + \left(\frac{d}{d\theta_4}J_{xx}\right)\cdot\dot{\theta}_4 + \left(\frac{d}{d\theta_5}J_{xx}\right)\cdot\dot{\theta}_5 + \left(\frac{d}{d\theta_6}J_{xx}\right)\cdot\dot{\theta}_6$$

As an example, the derivative of J_{A4} can be calculated as;

$$J_{A4} = \begin{pmatrix} c\theta_1\cdot s\theta_{23} \\ s\theta_1\cdot s\theta_{23} \\ -c\theta_{23} \end{pmatrix}$$

Using the formula;

$$dJ_{A4} = \left(\frac{d}{d\theta_1}J_{A4}\right)\cdot\dot{\theta}_1 + \left(\frac{d}{d\theta_2}J_{A4}\right)\cdot\dot{\theta}_2 + \left(\frac{d}{d\theta_3}J_{A4}\right)\cdot\dot{\theta}_3 + \left(\frac{d}{d\theta_4}J_{A4}\right)\cdot\dot{\theta}_4 + \left(\frac{d}{d\theta_5}J_{A4}\right)\cdot\dot{\theta}_5 + \left(\frac{d}{d\theta_6}J_{A4}\right)\cdot\dot{\theta}_6$$

The result is;

$$dJ_{a4} = \begin{pmatrix} \sin(\theta_1)\cdot\sin(\theta_2+\theta_3)\cdot d\theta_1 - \cos(\theta_1)\cdot\cos(\theta_2+\theta_3)\cdot d\theta_2 - \cos(\theta_1)\cdot\cos(\theta_2+\theta_3)\cdot d\theta_3 \\ -\cos(\theta_1)\cdot\sin(\theta_2+\theta_3)\cdot d\theta_1 - \sin(\theta_1)\cdot\cos(\theta_2+\theta_3)\cdot d\theta_2 - \sin(\theta_1)\cdot\cos(\theta_2+\theta_3)\cdot d\theta_3 \\ \sin(\theta_2+\theta_3)\cdot d\theta_2 + \sin(\theta_2+\theta_3)\cdot d\theta_3 \end{pmatrix}$$

Appendix C

Users Manual for the METUROBOT System using Developed MMI

C.1 Introduction

In this manual, it is intended to give a detailed and step-by-step explanation of how to operate METUROBOT, using developed MMI software. The manual is divided into 4 sections;

- Describing the METUROBOT,
- Starting system for operating the METUROBOT.
- Operating robot with controlling joints,

and,

- Operating robot with controlling the position of the tip point of the robot.

C.2 The METUROBOT

METUROBOT is a 6 degree-of-freedom industrial robot in CAD/CAM/ROBOTICS centre. It is designed and manufactured by the thesis students of Prof. Dr. Bilgin Kaftanoğlu. The robot is shown in figure C.1.

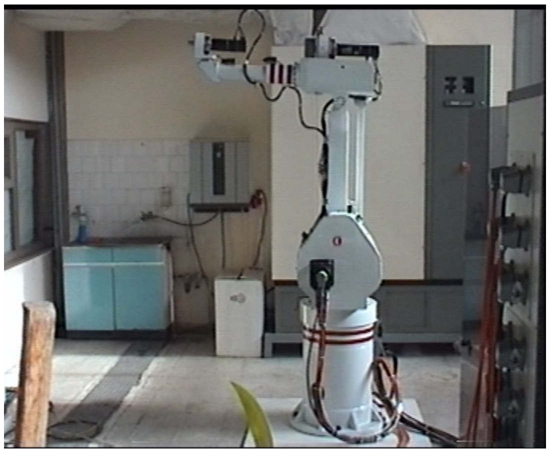


Figure C.1 METUROBOT

The robot has 6 AC servomotors and they are controlled by Delta-Tau motion control card. The schematic view of the robot is shown below (Figure C.2);

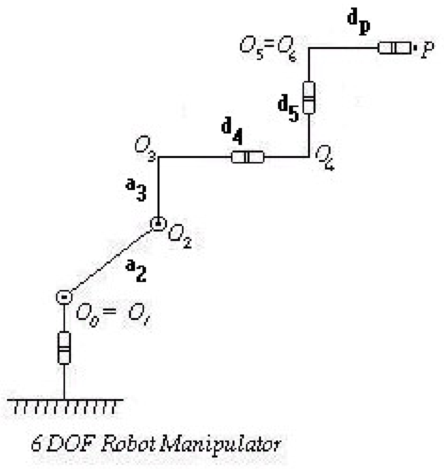


Figure C.2 Schematic View of the METUROBOT

P is the tip point of the robot. The length of the distance d_P can be adjusted for the end-effector used. In the MMI software, there is a menu for adjusting the values of the link lengths. The link lengths are in the following table (Table C.1):

Table C.1 Link Lengths

Link	Length (mm)
Base to O_0	1201
a_2	800
a_3	152,5
d_4	895
d_5	164,8
d_P	250

There are 6 joints and all these joints are revolute. In order to operate the robot in safely, there are limit switches on the joints to define the limits. It is not recommended to use the robot near the joint limits. Limits of the motors in terms of angles are in the following table, (Table C.2). Angle values are from the home position. Motors are controlled by motion control card. Position feedback is taken from the incremental encoders. Last column shows the encoder counts per revolution, i.e. 360 degrees is equal to 360,000 counts for encoder of motor 1, etc.

Table C.2 Limits of the Motors

Motor #	Min. Angle (deg)	Max. Angle (deg)	Encoder Counts per Revolution
1	-180	180	360.000
2	-10	10	585.000
3	-20	20	510.000
4	-160	160	510.000
5	-110	110	470.000
6	-180	180	360.000

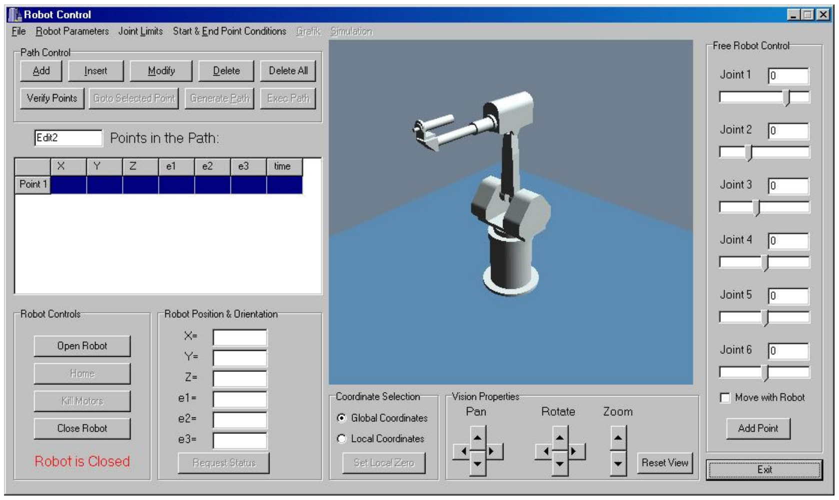
These limits restrict the movement of the robot, but it has a sufficient workspace around the robot

C.3 Starting the System

In this part, starting the whole system, robot and the software will be explained. More detailed users manual for installing the motion card, motion software and tuning the motors, can be found on the thesis of Oykun Eren.

The steps for starting the system are listed below;

1. Turn on the computer,
2. Run "Pewin32.exe" program and "MMI of METUROBOT" softwares, the screen view of the MMI software is shown below (Figure C.3). the shortcuts of these programs are on the desktop of the computer.



Robot is Closed

[✓ Global Coordinates] Local Coordinates

Joint 1: 0
Joint 2: 0
Joint 3: 0
Joint 4: 0
Joint 5: 0
Joint 6: 0

Figure C.3 Screen View of the MMI Software

3. Press the “Open Robot” button located in the “Robot Control” section of the software,
4. Press the “Kill Motors” button under the “Open Robot” button,
5. Then, plug the main socket (3-phase plug of the control cabinet) into the socket in the electric box on the wall and turn on the main power switch inside the box,
6. Check the workspace of the robot is clear of objects and personnel,
7. Turn on the “main switch” inside the cabinet,
8. Press the “start” button in the cabinet,
9. Press the “run” button,

The buttons to control the robot in the cabinet are shown below (Figure C.4).

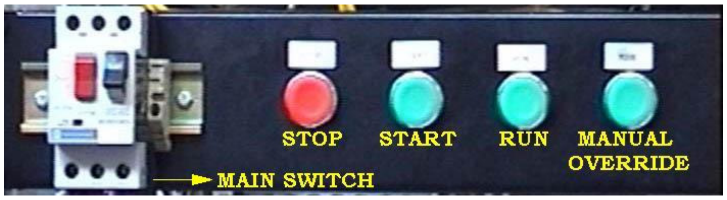


Figure C.4 Buttons in the Cabinet

10. Press the “Home” button on the software to hold the robot on home position.

The robot is now ready to operate.

C.4 Operating Robot with Controlling the Joints

The right side of the software is reserved for the joint control of the robot. When controlling joints from software, do not make big changes in a short time.

When the user changes the values of the joint angles from the slide bars, the robot view in the simulation window will move. If the user wants to move the METUROBOT, then checks the checkbox “Move with Robot” on the right bottom side of the software, and changes the values of the angles from the slide bars of the joints. The robot will move simultaneously with the simulation.

The important notice is that, if the robot simulation and METUROBOT are in different positions, the METUROBOT will suddenly move the position shown in the screen, when the checkbox is checked.

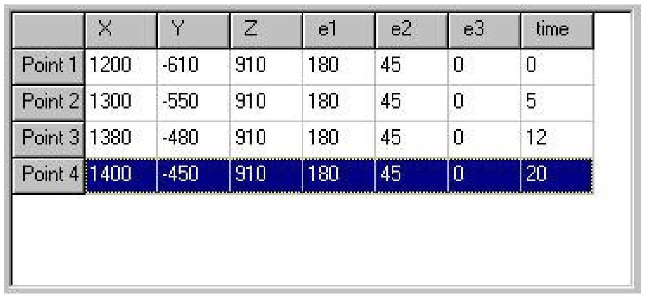
C.5 Operating Robot with Controlling the Pose of the End-Effector

The main aim of the MMI software is the controlling the pose of the end-effector of the robot. There is a list of the points on the left of the simulation window. User fills the list from “Add” (add point) button located on the top of the list. User can also modify or delete a point using the buttons. The points are inserted to the list with 7 parameters, x-y-z values on the space coordinates, e1-e2-e3 euler angles and time. Time is used for the path planning.

After filling the table, points are verified with the “Verify Points” button. If there is a point out of the workspace, the points are not verified. If all the points are verified, the point is shown on the simulation window by clicking the point row on the list. To move the METUROBOT to the point, press “Go To Selected Point” button when the point is selected from the list.

C.6 Path Generation with MMI software

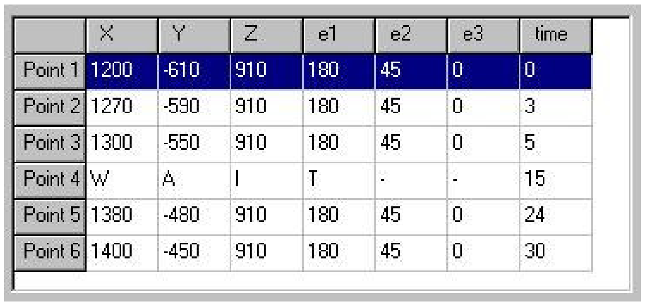
If the user wants to move the METUROBOT on the path, he lists the points on the path and presses the “Generate Path” button after verification of all the points. After the path is generated, joint limits (position, velocity and acceleration) checked, “graphics” and “simulation” menu items are enabled. User can observe the planned trajectory by pressing the “Simulate” menu. Example point listing for one path is in figure C.5. In this example, end-effector of the robot move from point 1 to point 4, in 20 seconds and touches the intermediate points at given times.



	X	Y	Z	e1	e2	e3	time
Point 1	1200	-610	910	180	45	0	0
Point 2	1300	-550	910	180	45	0	5
Point 3	1380	-480	910	180	45	0	12
Point 4	1400	-450	910	180	45	0	20

Figure C.5 Example points for the single path

The user can define more than one path, one after the other. In this case, user can put a waiting line and inputs the waiting time in the time section. Example points for 2 paths are in figure C.6



	X	Y	Z	e1	e2	e3	time
Point 1	1200	-610	910	180	45	0	0
Point 2	1270	-590	910	180	45	0	3
Point 3	1300	-550	910	180	45	0	5
Point 4	W	A	I	T	-	-	15
Point 5	1380	-480	910	180	45	0	24
Point 6	1400	-450	910	180	45	0	30

Figure C.6 Example points for the multi-path

When these points are entered, robot goes point 1 to point 3 in 5 seconds, then waits in the point 3 for 10 seconde and continues to the path. The end point is the point 6 and the total time of travel is 30 seconds with pause.