

DOKUZ EYLÜL UNIVERSITY
GRADUATE SCHOOL OF NATURAL AND APPLIED
SCIENCES

PC-BASED CONTROL OF HEXAPOD

by
Halil ŞAHBAZ

January, 2007
İZMİR

PC-BASED CONTROL OF HEXAPOD

**A Thesis Submitted to the
Graduate School of Natural and Applied Sciences of Dokuz Eylül University
In Partial Fulfillment of the Requirements for the Degree of Master of Science
in Mechanical Engineering, Machine Theory and Dynamics Program**

**by
Halil ŞAHBAZ**

**January, 2007
İZMİR**

M.Sc THESIS EXAMINATION RESULT FORM

We have read the thesis entitled “**PC-BASED CONTROL OF HEXAPOD**” completed by **HALİL ŞAHBAZ** under supervision of **PROF.DR. HİRA KARAGÜLLE** and we certify that in our opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.

.....
Prof. Dr. Hira KARAGÜLLE

Supervisor

.....
Assist.Prof.Dr. Zeki KIRAL

(Jury Member)

.....
Assist.Prof.Dr. Aysun BALTACI

(Jury Member)

.....
Prof.Dr. Cahit HELVACI

Director

Graduate School of Natural and Applied Sciences

ACKNOWLEDGEMENT

I am greatly indebted to my thesis supervisor Prof. Dr. Hira KARAGÜLLE for kindly providing guidance and interest throughout the development of this study.

I would like to thank to Research Assistant Levent MALGACA, Research Assistant Hasan ÖZTÜRK, Instructor Kemal VAROL from the Machine Theory and Dynamics Program and my colleague Burcu GÜNERİ, and my all friends, for their all kind of supports during the period of my study.

This thesis is a part of a research project (TÜBİTAK, Project number:104M373). Therefore, I would also like to thank to The Scientific & Technological Research Council of Turkey (TÜBİTAK) for the support to the project.

This thesis is dedicated to my father, my mother and my sister. I would like to thank them for their support, patience and sacrifice.

Halil ŞAHBAZ

PC-BASED CONTROL OF HEXAPOD

ABSTRACT

In this study, the aim is to control an hexapod with PC-based control. In this manner, a 6-DOF parallel manipulator known as hexapod used in the areas such as precise manufacturing and medicine is examined. For the experimental study, an hexapod is designed and manufactured. Simulations are done in VisualNASTRAN to determine actuator lengths of the hexapod. Actuators of the hexapod are linear step motors. In order to control the hexapod, an integrated VisualBASIC program which uses VisualNASTRAN is developed. The program runs simulation, takes simulation results as inputs and controls linear step motors via PC-based motion controllers. Point-to-point open loop control is applied. Positions of the hexapod in consequence of controlling are measured with a coordinate measuring machine (CMM) and results are presented. Furthermore, for another experimental study, a servo motor experimental rig is set with three AC servo motors which are actuators of a 3-DOF serial manipulator on which the hexapod will be attached. Simulations are done in VisualNASTRAN to determine actuator positions. VisualBASIC programs whose inputs are simulation results are developed to control servo motors according to different control methods and algorithms via PC-based motion control cards. Open loop and closed loop path tracking control are applied. Results are presented. Motors are successfully followed reference curves and give good responses regarding proposed algorithms.

Keywords: Hexapod, micro-positioning, PC-based motion control, step motor control, servo motor control.

HEGZAPODUN BİLGİSAYAR TABANLI KONTROLÜ

ÖZ

Bu çalışmada amaç bilgisayar tabanlı kontrol ile bir hegzapodu kontrol etmektir. Bu bağlamda, hassas üretim ve tıp gibi alanlarda kullanılan hegzapod olarak bilinen 6 serbestlik dereceli bir paralel manipülatör incelenmiştir. Deneysel çalışma için, bir hegzapod tasarlanmış ve imal edilmiştir. Hegzapodun tahrik elemanlarının uzunluklarını belirlemek için VisualNASTRAN'da simülasyonlar yapılmıştır. Hegzapodun tahrik elemanları adım motor sürücülü doğrusal motorlardır. Hegzapodun kontrolü için VisualNASTRAN'ı kullanan bir entegre VisualBASIC programı geliştirilmiştir. Program simülasyonu çalıştırır, simülasyon sonuçlarını girdi olarak alır ve bilgisayar tabanlı hareket kontrol üniteleri aracılığıyla adım motor sürücülü doğrusal motorları kontrol eder. Noktadan noktaya açık devre hareket kontrolü uygulanmıştır. Hegzapodun kontrol sonucundaki konumları, bir koordinat ölçüm makinesi ile ölçülmüş ve sonuçlar sunulmuştur. Ayrıca, diğer bir deneysel çalışma için üzerine hegzapodun bağlanacağı bir 3 serbestlik dereceli seri manipülatörün tahrik elemanları olan üç AC servo motor ile, bir servo motor deney düzeneği kurulmuştur. Tahrik elemanlarının konumlarını belirlemek için VisualNASTRAN'da simülasyonlar yapılmıştır. Servo motorların değişik kontrol yöntemleri ve algoritmalara göre bilgisayar tabanlı hareket kontrol kartları üzerinden kontrolü için, girdileri simülasyon sonuçları olan VisualBASIC programları geliştirilmiştir. Açık devre ve kapalı devre yörünge izleme kontrolü uygulanmıştır. Sonuçlar sunulmuştur. Motorlar önerilen algoritmalara göre hedef eğrileri başarıyla takip etmiş ve iyi cevaplar vermişlerdir.

Anahtar sözcükler: Hegzapod, mikro-konumlandırma, bilgisayar tabanlı hareket kontrolü, adım motor kontrolü, servo motor kontrolü.

CONTENTS

	Page
THESIS EXAMINATION RESULT FORM	ii
ACKNOWLEDGEMENTS	iii
ABSTRACT	iv
ÖZ	v
CHAPTER ONE – INTRODUCTION	1
1.1 Robot Manipulators	3
1.1.1 Parallel Manipulators	3
1.1.2 Serial Manipulators	6
1.2 Controllers and Actuators	7
1.2.1 Controllers	7
1.2.2 Actuators	7
1.2.2.1 Linear Step Motors	8
1.2.2.2 AC Servo Motors	9
1.3 Literature Review	10
CHAPTER TWO – INITIAL DESIGN OF THE HEXAPOD	13
2.1 Introduction	13
2.2 Initial Design	13
2.3 Kinematic Analysis	14
2.4 Kinetic Analysis	15
CHAPTER THREE – SELECTION OF APPROPRIATE ACTUATORS AND JOINTS	16
3.1 Introduction	16
3.2 Selecting Appropriate Linear Actuators	16
3.3 Selecting Appropriate Joints	18

CHAPTER FOUR –	
DESIGN AND MANUFACTURING OF THE HEXAPOD	21
4.1 Introduction	21
4.2 Design of the Hexapod	22
4.2.1 Generating 2D and 3D Drawings of Parts	22
4.2.2 Testing the Hexapod with VisualNASTRAN	23
4.3 Manufacturing of the Hexapod.....	27
4.4 Hexapod Control System.....	29
CHAPTER FIVE – SIMULATION AND CONTROLLING OF THE	
HEXAPOD.....	35
5.1 Introduction	35
5.2 Simulation and Controlling of the Hexapod.....	36
5.2.1 A Developed VisualBASIC Program for Simulation and Control.....	36
5.2.2 Simulation and Inverse Kinematic Analysis of the Hexapod.....	38
5.2.3 Controlling of the Hexapod	40
CHAPTER SIX – TEST RESULTS OF THE HEXAPOD	42
6.1 Introduction	42
6.2 Tests and Results	42
CHAPTER SEVEN–CONTROLLING OF AC SERVO MOTOR SYSTEMS. 56	
7.1 Introduction	56
7.2 Design and Inverse Kinematic Analyses of a 3-DOF Serial Manipulator	56
7.3 AC Servo Motor Experimental Rig.....	58
7.3.1 The Experimental Rig.....	58
7.3.2 Connections of Servo Motors and Drivers	60
7.3.3 Adjusting Servo Motor Parameters	63
7.4 Controlling of AC Servo Motors.....	65
7.5 Results of the Controlling Servo Motor Systems	78

CHAPTER EIGHT – CONCLUSIONS.....	80
REFERENCES.....	82
APPENDIX A – PROPERTIES OF ADLINK PCI 8132 AND PCI 8164	
MOTION CONTROL CARDS.....	88
A.1 Features of PCI 8132 and PCI 8164.....	88
A.2 Fundamental Commands of PCI 8132 and PCI 8164	90
APPENDIX B – ACCELERATION – DECELERATION SINUSOID.....	91
B.1 Definition of the Acceleration – Deceleration Sinusoid.....	91
B.2 Creating Samples of the Sinusoid by VisualBASIC.....	91
APPENDIX C– 2D MANUFACTURING DRAWINGS OF THE HEXAPOD. 93	
APPENDIX D – VISUALBASIC PROGRAM FOR SIMULATION AND	
CONTROLLING OF THE HEXAPOD	101
APPENDIX E – VISUALBASIC PROGRAM FOR CONTROLLING OF	
SERVO MOTOR SYSTEMS.....	113

CHAPTER ONE

INTRODUCTION

Controlling of robot manipulators in micron ranges is an important issue of developing technology. Micro-positioning robots have a wide variety of applications from surgery robots to space shuttles. These robots can precisely follow a path described with respect to the work. Working with more precise motions and making more precise machines decrease errors which might come into existence because of effects of standard machines and workers. By more precise manufacturing machines, dimensions of manufactured parts become more accurate. Machines which made by more accurate parts work with less errors. Thus, machines make much more accurate and reliable products. Parallel manipulators are used for micro-positioning. Best-known type of parallel manipulators is hexapod. Hexapods are suitable for micro-positioning, because hexapods have 6 degrees of freedom.

In this thesis, PC-based control of a 6-DOF parallel manipulator namely hexapod is comprised. In this manner, hexapod for micro-positioning and controlling of actuators of a serial manipulator on which the parallel manipulator will be attached at future works are considered. Actuators of the hexapod and the serial manipulator are linear step motors and brushless AC servo motors, respectively. These actuators are controlled by PC-based motion control cards. Point-to-point open loop control is applied to linear step motors, so to hexapod; and open loop and closed loop control is applied to servo motors for path following.

In chapter one, there is introductory knowledge about the thesis. In the second chapter, in order to determine minimum requirements of actuators and joints of hexapod, an initial design is considered. In the third chapter, supplied actuators and joints are presented. In the fourth chapter, the hexapod is designed and manufactured. And control system of the hexapod is created. In the fifth chapter, the hexapod is simulated according to manufactured hexapod and controlled with respect to simulation results with developed programs. In the following chapter, positions of the hexapod are measured and the results are presented. In the seventh chapter, an

experimental rig is set with actuators of the designed serial 3-DOF manipulator; different control algorithms are examined with developed programs. It is tried to find which control algorithm and gain coefficients are appropriate for the system.

Simulations are done with VisualNASTRAN (MSC Software Corp., 2006). In order for programming, VisualBASIC (MSDN, 2006) is used. 3D solid modelling and 2D manufacturing drawings are accomplished by I-Deas and SolidWorks (SolidWorks Corp., 2006) solid modelling programs. 3D models of designed serial manipulator are created with ABAQUS (Abaqus Inc., 2006) analysis program. An integrated analysis is performed.

This thesis is a part of a research project (TUBITAK, Project number:104M373). Some sections of the thesis are created by taking consideration of the project reports of Karagülle, H., Sarigül, S., Kırıl, Z., Varol, K., & Malgaca, L. (01 July 2006, 01 July 2007). The flow chart of the integrated analysis can be seen in Figure 1.1. Researches of the project is followed the flow chart.

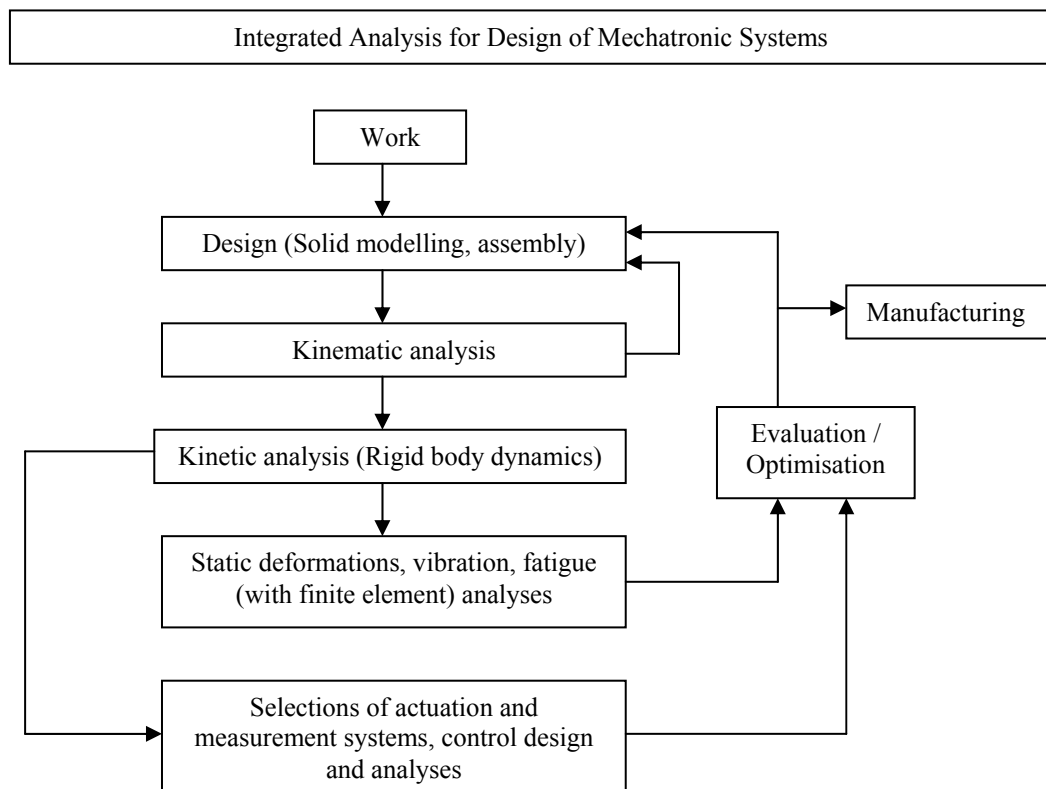


Figure 1.1 The flow chart of the integrated analysis.

This thesis particularly contains “selection of actuation and measurement systems, control design and analyses” part of the flow chart (Fig. 1.1).

Because of examining robot manipulators with PC-based motion control, the explanations about them will be useful.

1.1 Robot Manipulators

Robots are used in many varied applications, from welding to surgery. A machine has to be able to do followings for being a robot: getting information from surroundings; doing something physical, such as moving or manipulating objects; reprogrammable, it can do different things; function autonomously and / or interact with human beings. Robot name comes from a play whose name is Rossum’s Universal Robots of Czech writer Karel Capek, published in 1921 (McKerrow, 1991). Robotic Institute of America defines a robot as: “A robot is a re-programmable multi-functional manipulator designed to move material, parts, tools, or specialized devices, through variable programmed motions for the performance of a variety of tasks.” (Tsai, 1999).

Robot manipulators are divided into parallel robot manipulators and serial robot manipulators with respect to their kinematic chains. If manipulators having an open kinematic chain structure, are called serial manipulators; having a closed kinematic chain, are called parallel manipulators.

1.1.1 Parallel Manipulators

A parallel manipulator is a closed chain mechanism which has two platforms (base and moving platform) connected together by at least two independent kinematic chains. Moving platform is connected to the fixed base by several limbs or legs. Typically the number of limbs is equal to the number of degrees of freedom

such that every limb is controlled by one actuator and all the actuators can be mounted at or near the fixed base (Tsai, 1999).

The parallel manipulators have some advantages such as higher stiffness and greater load-to-weight ratio because each actuated leg has to carry only a part of payload. This is quite energy efficient and the robot can handle heavy loads. In situations, where the accuracy, high speed and stiffness are more important than workspace, parallel manipulators can be alternative to serial ones whereas reduced workspace, difficult mechanical design, more complex direct kinematics and complex control algorithms are the main disadvantages of them.

Parallel manipulators came into existence when Gough & Whitenhall (1962) devised a six-linear jack system as a universal tire-testing machine. Stewart (1965) designed an aircraft simulator with a platform and six actuators. Thus, 6-DOF parallel manipulators namely hexapods, are also known as Stewart – Gough platforms. A schematic view of an hexapod is given in Figure 1.2.

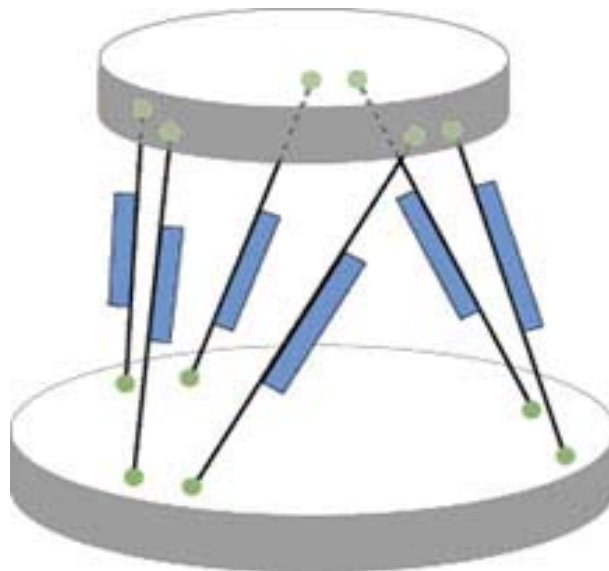


Figure 1.2 A schematic view of an hexapod.

A hexapod producer company called PI (Physik Instrumente Co., 2006) is announces advantages of hexapods as:

- Lower inertia,
- Better dynamic behaviour,
- Smaller package size,
- Higher stiffness,
- No accumulation of position errors,
- Reduced run out errors,
- No moving cables: better repeatability and reliability.

Application areas of hexapods are presented from PI Company (Physik Instrumente Co., 2006) as following:

- Alignment and tracking of optics, electron beams, lasers, etc.,
- Satellite testing equipment,
- Surgical robots,
- Micromachining,
- Micromanipulation (life sciences),
- X-ray diffraction measurements,
- Semiconductor handling systems,
- Tool control for precision machining and manufacturing,
- Fine positioning of active secondary mirror platforms in astronomical telescopes.

In Figure 1.3, hexapods in surgical applications can be found.

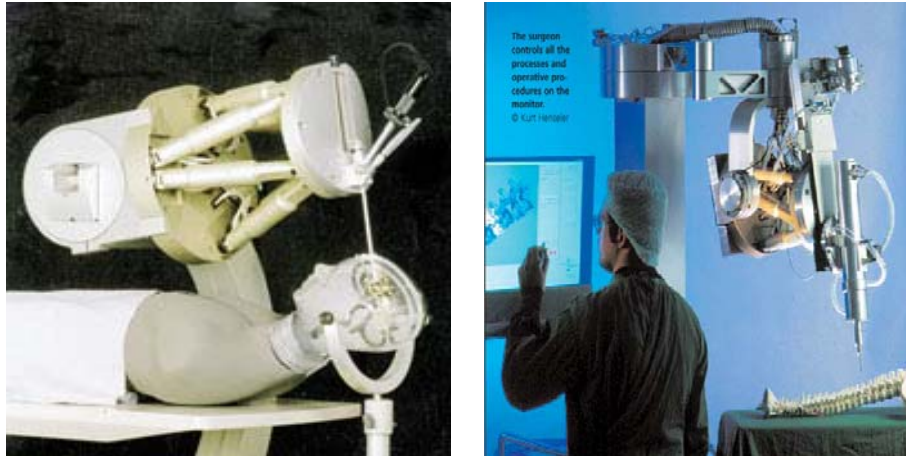


Figure 1.3 Hexapods in surgical applications.

1.1.2 Serial Manipulators

Serial manipulators are used to be the most common type of robot manipulators. They have open kinematic chain and serially connected links. This type of manipulators has the advantages of to be able to achieve high velocities and accelerations because the end-effector moves generally faster than the actuated links and their workspaces are relatively high. They have also larger workspace than parallel manipulators. Serial manipulators do not have energy efficiency because all actuators have to actuate other links and actuators through the end point. A serial type robot manipulator is presented in Figure 1.4.

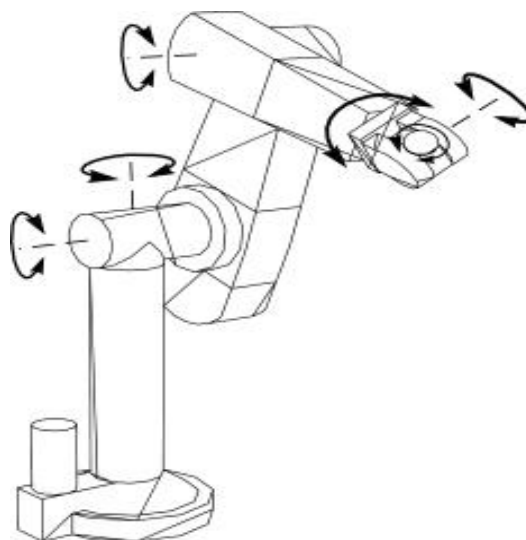


Figure 1.4 A serial manipulator (Puma type)

1.2 Controllers and Actuators

1.2.1 Controllers

Motors can be controlled by PC-based, stand-alone or hybrid systems. A hybrid system is a sum of PC-based and stand-alone units. In this thesis, motors are controlled by ADLINK PCI 8132 and PCI 8164 motion control cards (Adlink Inc., 2006) as PC-based motion control. Comprehensive knowledge about PCI 8132 and PCI 8164 motion control card can be found in Appendix A.

PC-based systems require a PC to run all time when the process is running. Stand-alone units do not require a PC to run. However, they might use a PC for programming. Hybrid units contain a PC and a stand-alone unit. Hybrid units run without PCs but use PCs to add more functions. A comparison table is compiled in Table 1.1 (Custom Solutions Inc., 2006).

Table 1.1 Comparison of PC-based, stand-alone and hybrid systems.

	PC-based	Stand-alone	Hybrid
Reliability	Low (Because of PC and software failures)	High (Because they do not require PCs)	The same as stand-alone units
Cost	Low	High	Highest
Power consumption	High (Because of PCs)	Low	Higher (Because they are a sum of two other systems)
Equipment size	High	Low	Highest
Noise	High (Because of PCs)	Low	The same as stand-alone units.

1.2.2 Actuators

In this study, electrical motors such as linear step motors for actuation of the hexapod and AC servo motors for actuation of the serial manipulator are controlled.

An electrical motor is a machine which converts electric energy into mechanical energy. When an electric current is passed through a wire loop which is in a magnetic field, the loop will rotate and the rotating motion is transmitted to a shaft, providing useful mechanical work. A traditional electric motor consists of a conducting loop that is mounted on a rotational shaft. The electrical current fed in by carbon blocks called brushes and enters the loop through two slip rings. The magnetic field around the loop supplied by an iron core field magnet causes the loop to turn when current is flowing through it.

1.2.2.1 Linear Step Motors

Step motors are driven by a train of electrical pulses. Pulse train results rotational speed. The stator is wound as two separate coils, which produce magnetic fields offset angularly by half a rotor pole. These coils are pulsed alternately to produce a rotating magnetic field. The rotor which is polarized with alternating north and south poles aligns itself with this field, and rotates with it. Each pulse turns the rotor through a fixed angle (one step) and consequently, angular position change is proportional to the number of pulses. Accurate open loop control of the rotational velocity is achieved by controlling the pulse rate. Step motor can slip during rapid acceleration of high inertia loads, thus, for accurate open loop control, the pulse frequency has to be varied during times of acceleration. Feedback control is used with step motors in situations where accuracy is required (McKerrow, 1991).

Advantages of step motors are as the followings (Baldor Co., 2006):

- Step motors can be simply controlled,
- Step motors have good results at constant loads,
- Step motors have good results at positional accuracy,
- Costs of step motors are low.

Disadvantages of step motors are as the followings (Baldor Co., 2006):

- Step motors can lose steps,
- Step motors are not good at varying loads,
- Step motors have energy inefficiency,
- Step motors can have resonance problems,
- Step motors have large motor sizes.

Linear step motors change motion of output shafts from rotational motion to linear motion, due to a spindle-nut mechanism. Thus, rotational motion is changed to linear motion, basically. Linear motors are more useful than hydraulic-pneumatic systems which require pumps and compressors and conditioning the air. Because linear motors are driven by electricity; linear motors provide direct linear motion without the potential complications associated with pneumatic and hydraulic systems, without mechanical linkages such as ball screws or rack-and-pinion systems and without noise, oil and large working environments.

1.2.2.2 AC Servo Motors

This kind of motor works with the electrical current flow in the laminate core loop. The speed of AC induction motors is set roughly by the motor construction and the frequency of the current. In order to control the motor speed, it is necessary to use a mechanical transmission. The rotor circuit can be connected to various external control circuits to obtain greater flexibility. In recent years the AC servo has taken over from the DC servo as the standard drive. These modern motors give higher power output and are almost silent in operation. As they have no brushes, they are very reliable and require almost no maintenance in operation.

AC servo motors divided into synchronous and asynchronous (induction) motor. A synchronous motor is basically the same as an asynchronous motor. However it is slightly different from rotor construction. The rotor construction enables a

synchronous motor to rotate at the same speed (in synchronisation) as the stator field. (Yaskawa, 2002).

Advantages of servo motors are (Baldor Co., 2006):

- Servo motors have high performance,
- Servo motors have high speeds available with specialized controls,
- Servo motors have wide variety of components,
- Small size.

Disadvantages of servo motors are (Baldor Co., 2006):

- High performances of servo motors are limited by controls and controllers,
- High speed torque of servo motors are limited by commutator or electronics,
- Servo motors have higher costs compared to step motors.

1.3 Literature Review

6-DOF parallel manipulators were firstly used by Gough & Whitehall (1962) as a tire-testing machine. The testing machine was consisted of a six-linear jack system. Stewart (1965) designed a 6-DOF parallel manipulator for the usage of flight simulator. A systematic study of kinematic structures of parallel manipulators was made by Hunt (1983). Since then, parallel manipulators have been taking interests of researchers.

Wendlandt & Sastry (1994) examined a Stewart platform for endoscopy in order to design and control. The researchers aimed the platform to follow a circled path. McInroy (1999) investigated controlling of hexapods by dynamic modelling. Base accelerations were included and the model was experimentally verified. A comprehensive literature review study was made by Dasgupta & Mruthyunjaya (2000). The study contained all topics about hexapods and researches related to these

topics made until the publication year. An important study about parallel manipulators was the investigating new kinematic structures for parallel manipulators (Gao, Li, Zhao, Jin & Zhao, 2002). In that study, researchers developed new types of composite links. Joint types related to degree of freedom were presented. Alizade & Bayram (2004) classified parallel manipulators according to their platform types and connections between them. In another study, active vibration control of an hexapod was achieved with sensitivity weighted linear quadratic Gaussian (SWLQG) controller (Hauge & Campbell, 2004). Inverse kinematics, forward kinematics, error analysis and workspace evaluation were examined in the paper of Jelenkovic, Jakobovic & Budin (2004). Drive singularities of parallel manipulators were investigated by Ider (2005). Kim, Cho & Lee successfully controlled a 6-DOF parallel manipulator, namely hexapod, with respect to robust nonlinear control. The researchers considered friction of each actuator because friction may degrade control performance. In this manner, in order to determine friction values, a friction estimator was used.

There are many researches about controlling of servo motors and step motors. Van de Straete, Degezelle, De Schutter & Belmans (1998) have investigated servo motor selection criterions for mechatronic applications. Servo motors were modelled and simulated by using their mechanical and electrical properties in the paper of Dulger, Kirecci, & Topalbekiroglu (2001). By fuzzy logic control, an ultrasonic motor (Bal, Bekiroglu, Demirbas, & Colak, 2004) and a DC servo motor (Khongkoom, Kanchanathep, Nopnakeepong, Tamthong, Tunyasirirut, & Kagawa, 2000), (Lin, 1994) and (Lu, 1997) were controlled. Dandil, Gokbulut, & Ata (2004) and Lin & Wai (1998) investigated, respectively, asynchronous motors and synchronous motors with hybrid controllers which were adjusted by proportional-integral (PI) controllers via neural networks. Servo motors were controlled by various type control methods such as H_∞ robust control (Ximei & Qingding, 2005), adaptive fuzzy sliding-mode (Lin & Chiu, 1998), variable structure approach (Hashimoto, Yamamoto, Yanagisawa, & Harashima, 1988), micro-processor based robust control (Tzou & Wu, 1990) and learning approach (Han, Kim, Ha, Lee, & Park, 1995). Lin, Jan, Hwang, & Tsai (2003) studied kinematic analyses of hexapods

and controlling of AC servo motors which were used as actuators of hexapods in their study. Controlling of AC servo motors were achieved by using an estimator which estimated the rotor position as feedback (Yoneya, Yoshimaru, & Togari, 2000). Comparison of different servo motor drivers can be found in the study of Yamamoto & Shinohara (1996). Grimbleby (1995) and Crnosija, Adjukovic, & Kuzmanovic (1999) made studies on closed loop control algorithms of step motors and controlled step motors by using these algorithms. Mort, Abbod, & Linkens (1995) compared step motors which controlled by open loop control with proportional-integral (PI) controllers, and DC servo motors which controlled by closed loop fuzzy logic control. These two types of motors and methods gave good responses.

Studies about PC-based control systems are as followings: Path tracking of a 3-DOF CNC machine with circular and linear interpolation was examined; and tracking error was desired to decrease to minimum (Yang, & Hong, 2001). Ku, Larsen, & Cetinkunt (1998) used PC-based motion control cards for nano-positioning diamond machining tools. They also controlled the system by neural network. Noorani (1990) dealt with controlling of a 6-DOF robot manipulator actuated by step motors by giving position and orientation inputs via a computer.

CHAPTER TWO

INITIAL DESIGN OF THE HEXAPOD

2.1 Introduction

There are a lot of different structures of hexapods. After reviewing the literature, some different structures were created manually in VisualNASTRAN, to examine which type is optimal with respect to joint positions. In this chapter, different joint positions are tried and kinematic and kinetic analyses are performed in VisualNASTRAN.

2.2 Initial Design

In order to test different joint positions and to find characteristics of actuators and joints, a basic model is created (Karagülle et al., 2006). The model is shown in Figure 2.1.

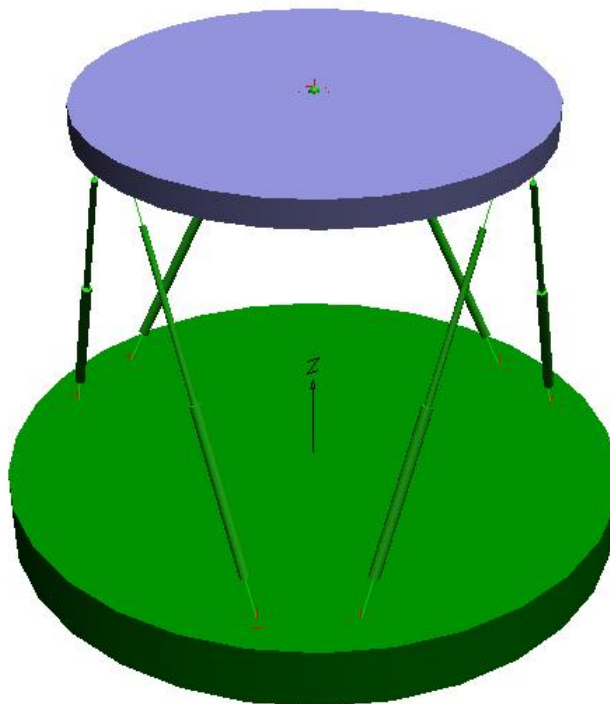


Figure 2.1 Basic model of hexapod

The basic model consists of a movable upper platform, a fixed lower platform and linear actuators. There are coordinates on the joint nodes on the platforms. Coordinates are assigned due to joint positions and linear actuators are created as constraints between coordinates on the upper platform and coordinates on the lower platform.

A world coordinate system is placed into the centre of lower platform of hexapod. For the design in Figure 2.1, diameter of the lower platform is 250 mm, diameter of the upper platform is 348 mm. Joints on the lower platform are placed 17.5 mm above with respect to world coordinate system and on a circle whose diameter is 200 mm; and the angle between closest joints is 20 degrees. Joints on the upper platform are placed 7.5 mm below of the centre of upper platform in the z direction and on a circle whose diameter is 298 mm; and the angle between joints is 60 degrees.

2.3 Kinematic Analysis

A VisualBASIC program is developed to create “prescribed motion”. V_x , V_y , V_z , W_x , W_y , and W_z values are assigned as prescribed motion. These are created with the VisualBASIC program with respect to a velocity sinusoid given in Appendix B (Karagülle et al., 2006). Inputs of the program are:

$$\begin{aligned} t_p &= 3: t_l = 0.2 * t_p: n = 21 \\ x_a &= 0: y_a = 0: z_a = 0.23: thx_a = 0: thy_a = 0: thz_a = 0 \\ x_b &= 0: y_b = 0.03: z_b = 0.2: thx_b = 10: thy_b = 0: thz_b = 0 \end{aligned}$$

Where, t_p (s) is the period of motion, t_l (s) is acceleration and deceleration time, n is number of samples, x_a , y_a , z_a , thx_a , thy_a , thz_a are components of the initial position and x_b , y_b , z_b , thx_b , thy_b , thz_b are components of the final position of the centre of the upper platform. Units of distances and rotations are meters and degrees, respectively.

It is desired the upper platform to move (0, 0, 0.23; 0, 0, 0) world coordinate point to (0, 0.03, 0.2; 10, 0, 0) world coordinate point in 3 seconds. 0.6 seconds is

acceleration time and 0.6 seconds is deceleration time and 1.8 seconds is constant velocity motion time (Karagülle et al., 2006).

After inserting prescribed motion components into the program, forces of the actuators are selected as zero. Thus, the upper platform moves only with velocity inputs. Meters are assigned to linear actuators to measure actuator lengths. Meters give lengths of actuators while the program is running. Meter values are saved for kinetic analysis in a file. Data of meters from this file can be taken by a VisualBASIC program. By comparing the values, maximum actuator length change is found as 47.58 mm.

2.4 Kinetic Analysis

In the kinetic analysis, “prescribed motion” option of the upper platform is disabled in VisualNASTRAN. Actuator lengths are assigned with values which have been saved after kinematic analysis. $F_x = -10$ N, $F_y = -10$ N, $F_z = -100$ N force inputs, and $T_x = 1$ Nm, $T_y = -1$ Nm, $T_z = 0$ Nm torque inputs are applied to the centre of the upper platform. Duration of the motion is taken 3 s and number of samples is 21 likewise the kinematic analysis. Constraint tension meters are created on the linear actuator constraints to measure actuator forces. Meters read the forces which occurred when the motion progresses. After the motion is stopped, maximum actuator force is found as 54.563 N with a subroutine of the VisualBASIC project which is developed (Karagülle et al., 2006).

Minimum requirements of linear motors and spherical joints are determined. According to results of the initial design, linear actuators have to have minimum 47.58 mm stroke and minimum 54.563 N force requirements. Spherical joints must have big tilt angle and high precision. These results are used while searching appropriate linear motors and joints.

CHAPTER THREE

SELECTION OF APPROPRIATE ACTUATORS AND JOINTS

3.1 Introduction

Choosing appropriate actuators and joints is an important part of creating an hexapod; because, active parts of an hexapod are linear motors and joints. Other parts are rigidly connected. In this chapter, choosing procedure of actuators and joints and properties of them are presented.

3.2 Selecting Appropriate Linear Actuators

Linear motors and joints were searched over the world and correspondences were made with companies which have web sites. Products were compared in terms of minimum stroke and minimum force requirements found from kinematic and kinetic analyses of the initial design (see Chapter 2). The values are 47.58 mm for minimum stroke and 54.563 N for minimum force requirements of linear actuators. They were also compared in terms of containing feedback devices, dimensions, precisions, velocities, weights, costs and delivery times. Suitable linear motors of companies which reply inquiries are presented Table 3.1.

After the comparing process, 4000 pulses/rev precision encoders attached 43K4U – 05 – 032ENG type linear motors of HSI Company (HSI Co., 2006) are supplied. One of the linear motors is shown in Figure 3.1. Technical properties of them are given in Table 3.2. The dimensions of the motors are presented in Figure 3.2 and additional knowledge about dimensions is given in Table 3.3. In addition, linear motors of PI Company (PI Co., 2006) might be also a good decision for future works.

Table 3.1 Linear motor companies and suitable linear motors.

	Stroke (mm)	Force (N)	Precision (μm)	Velocity (mm/s)	Dimension (mm)	Weight (g)	Type	Cost (USD)
PI	50	60	0.5	40	27 x 196	650	DC servo	2900
Ultra Motion	50.8	335	10	80	42x42x135	-	Stepper motor	730
Oriental	40	100	5	30	42x42x180	800	Stepper motor	1500
HSI	50.8	220	1.5	76	43x43x162	241	Stepper motor	120
Intelidrives	-	90	12	-	-	-	3 phase brushless	1700-2000
Linmot	100	122	100	4000	227+	740+ 460	AC servo	-
Iai	50	78.4	20	165	42x47x243	600	AC servo	-
Servoram	54	105	-	1370	84x84x182	7000	Servoram	5244

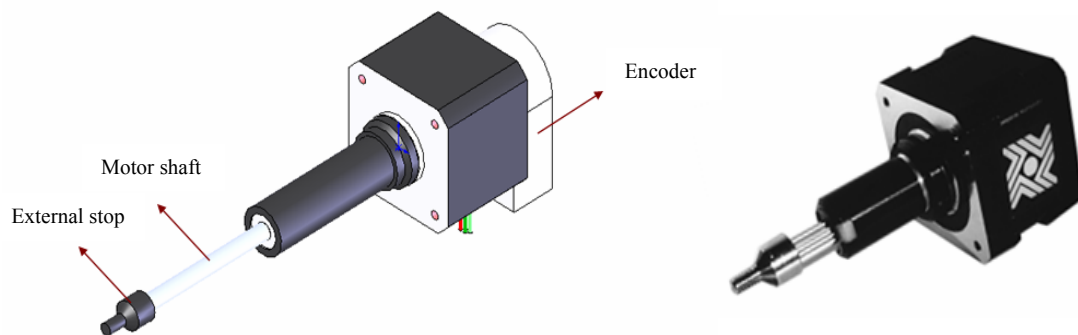


Figure 3.1 43K4U – 05 – 032ENG type linear motor of HSI Company.

Table 3.2 Technical properties of 43K4U – 05 – 032ENG type linear motor.

Connection type	Operating voltage (VDC)	Power (W)	Step angle (deg)	Precision (μm)	Stroke (mm)	Force (N)	Velocity (mm/s)
Bipolar	5	7	0.9	1.5	50.8	220	76

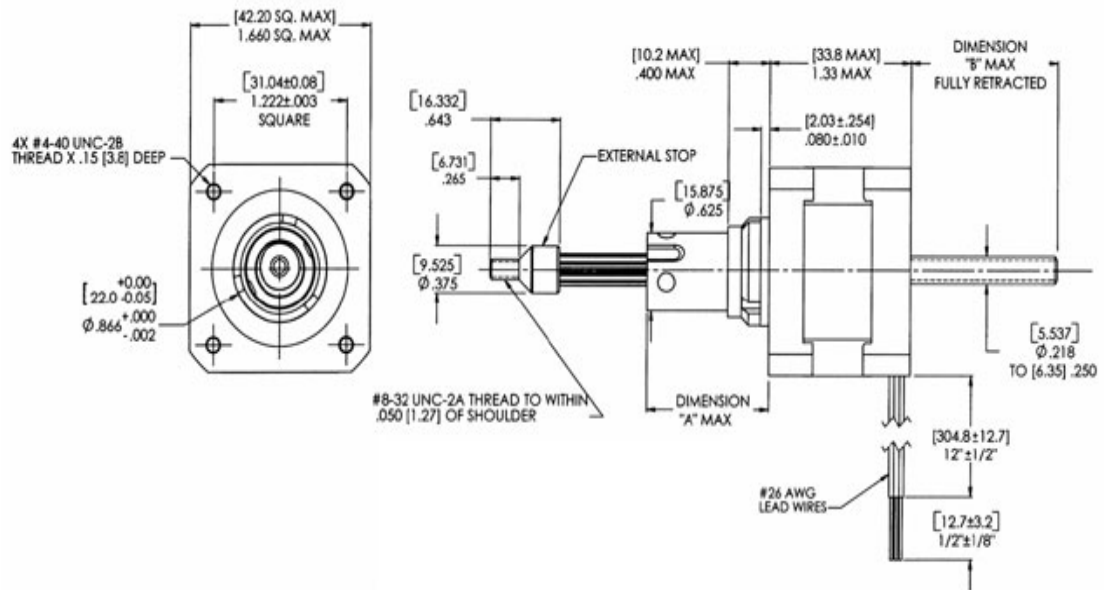


Figure 3.2 Dimensions of 43K4U – 05 – 032ENG type linear motor of HSI Company.

Table 3.3 Additional dimensions of 43K4U – 05 – 032ENG type linear motor of HSI Company.

Stroke Inch (mm)	Dimension "A" Inch (mm)	Dimension "B" Inch (mm)
2.00 (50.8)	2.28 (57.9)	1.66 (42.2)

3.3 Selecting Appropriate Joints

As a result of comparing spherical joints which are determined by searching the joint suppliers all over the world, twelve items of SRJ008C type 2.5 μm precision spherical joints, shown in Figure 3.3, of Hephaist Company (Hephaist Co., 2006) are supplied. Technical properties and dimensions are given in Table 3.4.

For investigating effects of different joint types over the precision at future works, twelve items of SSF.00.06 type spherical joints, shown in Figure 3.4, of Schaublin Company (Schaublin SA Co., 2004) from RS Company (RS Co., 2006) and six items of U5-13 161 type universal joints, shown in Figure 3.5, of Lenze Company (Lenze Co., 2006) are provided. Technical properties of the spherical joints and the universal joints are presented in Table 3.5 and Table 3.6, respectively.

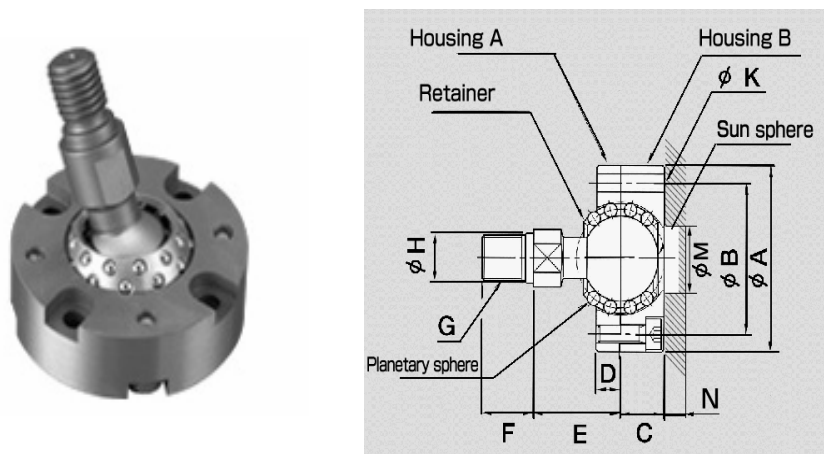


Figure 3.3 SRJ008C type spherical joint of Hephaist Company.

Table 3.4 Technical properties and dimensions of SRJ008C type spherical joint of Hephaist Company.

A	B	C	D	E	F	G	H	K	M	N	Force (N)	Weight (kg)
30	24	7	4	16	12	M5×0.5	5.5	3.4	11	2	540	0.06

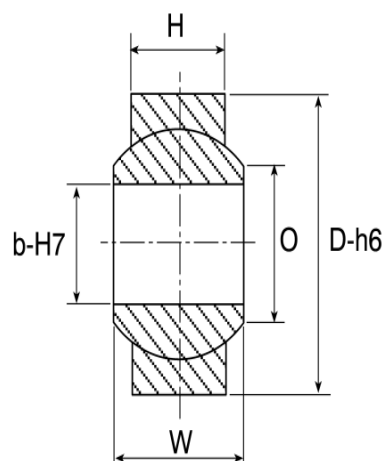


Figure 3.4 SSF.00.06 type spherical joint.

Table 3.5 Properties of SSF.00.06 type spherical joint.

Dimensions (mm)					Static force (N)	
b	D	H	O	W	Radial	Axial
6	18	6.75	8.96	9	980	240



Figure 3.5 U5-13 161 type universal joint.

Table 3.6 Properties of U5-13 161 type universal joint.

Nominal Torque (Nm) (200 rev/min, for 10° max.)	d (mm)	D (mm)	C (mm)	L1 (mm)	L2 (mm)	Weight (kg)
22	10	22	12	24	48	0.10

CHAPTER FOUR

DESIGN AND MANUFACTURING OF THE HEXAPOD

4.1 Introduction

Parts of the hexapod except from standard parts such as actuators and spherical joints are designed and manufactured. In this chapter, design and manufacturing of parts of the hexapod are presented. Control and driver systems of the hexapod are also set; and these systems are carried out. Simulation view of the hexapod is shown in Figure 4.1. Manufactured hexapod is given in Figure 4.2.

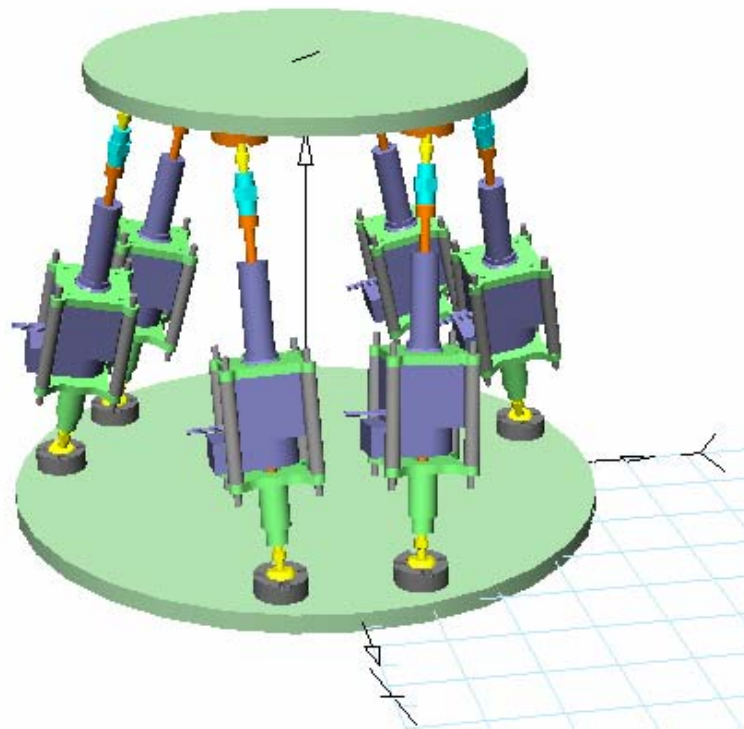


Figure 4.1 Simulation view of the hexapod.



Figure 4.2 Manufactured hexapod.

4.2 Design of the Hexapod

4.2.1 *Generating 2D and 3D Drawings of Parts*

Parts of the hexapod except from standard parts such as linear actuators and spherical joints are designed. Linear motor and spherical joint dimensions which are in Chapter 3 are taken as the base for design. Design of the parts is done with I-Deas solid modelling program. The parts are saved in “iges” format. Modifications of the parts are made with SolidWorks solid modelling program, if needed. Solid models of standard parts are also created for simulation (Karagülle et al., 2007).

Firstly, 3D solid models of all parts including linear motors and spherical joints are created. These parts are saved into “igs” extended files (Karagülle et al., 2007). Then “iges” files are imported manually into VisualNASTRAN to test joint locations on the upper and lower platforms. The model is constructed manually in the program (Fig. 4.1). Joint locations on the upper platform are not changed. However, joint locations on the lower platform are changed until the parts are not collided in the simulation when the hexapod goes to its limit points. The angle between closest joint locations on the lower platform is designated. After that, details of the platforms are determined. Thus, the parts are designed and 3D models of the parts are created.

2D manufacturing drawings are generated from the 3D solid parts in I-Deas. These drawings are presented in Appendix C (Karagülle et al., 2007).

4.2.2 Testing the Hexapod with VisualNASTRAN

“iges” formatted files of all parts which have been created of the hexapod are imported into the program. The parts are renamed. The original names and the VisualNASTRAN names of parts are in Table 4.1.

Table 4.1 Original names and VisualNASTRAN names of the parts.

Original names	VisualNASTRAN names
Lower platform	pa
Base of spherical joint on the lower platform	sa
Shaft of spherical joint on the lower platform	lsa
Lower connection part	lca
Base of linear motor	lma
Encoder	lea
Upper connection part	lda
Stud bolt	sp
Shaft of linear motor	lmb
Joint connection part	lcb
Shaft of spherical joints on the upper platform	lsb
Base of spherical joints on the upper platform	sb
Upper platform	pb

Coordinates which will connect parts to each other are located onto the parts. Coordinate locations with respect to body coordinates of each body are given in Table 4.2. In the table, rj notation is used for rigid joint, sj notation is used for spherical joint and lm notation is used for linear motor. Coordinates on the lower platform are changed regarding the angle between closest joints. This angle (φ_b) is changed 20 deg. to 30 deg., to test colliding of parts if occurs. Coordinates of “pa” and “pb” is created by a developed Matlab program which is below:


```

% b means base platform, p means upper platform
clc, clear
Db = 348; Dp = 250; Rt = 25;
Rb = (Db-2*Rt)/2; Rp = (Dp-2*Rt)/2;
fib = 30*pi/180; fip = 60*pi/180;      % joint angles
b1x = Rb*cos(fib/2); b1y = Rb*sin(fib/2);
b2x = Rb*cos(120*pi/180-fib/2); b2y = Rb*sin(120*pi/180-fib/2);
b3x = Rb*cos(120*pi/180+fib/2); b3y = Rb*sin(120*pi/180+fib/2);
b4x = Rb*cos(240*pi/180-fib/2); b4y = Rb*sin(240*pi/180-fib/2);
b5x = Rb*cos(240*pi/180+fib/2); b5y = Rb*sin(240*pi/180+fib/2);
b6x = Rb*cos(-fib/2); b6y = Rb*sin(-fib/2);

p1x = Rb*cos(fip/2); p1y = Rb*sin(fip/2);
p2x = Rb*cos(120*pi/180-fip/2); p2y = Rb*sin(120*pi/180-fip/2);
p3x = Rb*cos(120*pi/180+fip/2); p3y = Rb*sin(120*pi/180+fip/2);
p4x = Rb*cos(240*pi/180-fip/2); p4y = Rb*sin(240*pi/180-fip/2);
p5x = Rb*cos(240*pi/180+fip/2); p5y = Rb*sin(240*pi/180+fip/2);
p6x = Rb*cos(-fip/2); p6y = Rb*sin(-fip/2);

```

Parts are connected after coordinate settling (Table 4.2). The table is arranged according to the order of connections of assembly. Firstly lower platform is fixed to the ground and upper platform is located on (0, 0, 0.4) point ($z = 400\text{mm}$) with respect to world coordinates. Bases of spherical joints (1 to 6) are rigidly connected on the lower platform with respect to Table 4.2. Then parts named *lsa*, *lca*, *lma* and *lea*, *lda* and *sp* are rigidly connected, respectively; and five copies are created. After that *lmb*, *lcb* and *lsb* are rigidly connected, respectively; and five copies are created. Bases of other spherical joints (1 to 6) are rigidly connected on the upper platform with respect to Table 4.2. This step is demonstrated in Figure 4.3.

Connecting bodies are generated between the centres of the spherical joints on lower platform e.g. *sal-sl* and the centres of spherical joints on the upper platform e.g. *sbl-sl*. Thus, orientations of axes are found. The orientations, for $\varphi_b = 30$ deg, are in Table 4.3. These values are written into the orientations properties boxes of the

bodies of six linear motors (*lma*) and six shafts of linear motors (*lmb*). Thus axes orientations are done.

Table 4.2 Coordinate locations of the hexapod for one axis and for $\varphi_b = 30$ deg.

Part	Coordinate	X ¹⁾	Y ¹⁾	Z ¹⁾	Rx ¹⁾	Ry ¹⁾	Rz ¹⁾
Pa	Pa-fix	0	0	0	0	0	0
	Pa-j1	143.922948	38.564037	5.5-1=4.5	0	0	0
	Pa-j2	-38.564037	143.922948	5.5-1=4.5	0	0	0
	Pa-j3	-105.358910	105.358910	5.5-1=4.5	0	0	0
	Pa-j4	-105.358910	-105.358910	5.5-1=4.5	0	0	0
	Pa-j5	-38.564037	-143.922948	5.5-1=4.5	0	0	0
	Pa-j6	143.922948	-38.564037	5.5-1=4.5	0	0	0
Sa	Sa1-j1	0	0	0	0	0	0
	Sa1-s1	0	0	7	0	0	0
Lsa	Lsa1-s1	0	0	28	0	180	0
	Lsa1-c0	0	0	8	0	180	0
Lca	Lca1-c0	0	0	40	0	180	0
	Lca1-c11a	21.566	-21.566	0	0	180	0
	Lca1-c12a	21.566	21.566	0	0	180	0
	Lca1-c13a	-21.566	21.566	0	0	180	0
	Lca1-c14a	-21.566	-21.566	0	0	180	0
Sp (4 copies)	Sp1-c11a	0	0	27	0	0	0
	Sp1-c11b	0	0	-27	0	0	0
	Sp1-c12a	0	0	27	0	0	0
	Sp1-c12b	0	0	-27	0	0	0
	Sp1-c13a	0	0	27	0	0	0
	Sp1-c13b	0	0	-27	0	0	0
	Sp1-c14a	0	0	27	0	0	0
	Sp1-c14b	0	0	-27	0	0	0
Lma	Lma1-c1	0	0	33.2	0	0	0
	Lma1-c2	0	0	0	0	0	0
	Lma1-l1	0	0	-56.9	0	0	0
Lea	Lea1-c1	0	0	0	0	0	0
Lda	Lda1-c2	0	0	0	0	0	0
	Lda1-c11b	21.566	-21.566	0	0	0	0
	Lda1-c12b	21.566	21.566	0	0	0	0
	Lda1-c13b	-21.566	21.566	0	0	0	0
	Lda1-c14b	-21.566	-21.566	0	0	0	0
Lmb	Lmb1-l1	0	0	133.9	0	0	0
	Lmb1-c1	0	0	142.9932	0	0	0
Lcb	Lcb1-c1	0	0	2.3	0	0	0
	Lcb1-c2	0	0	22	0	0	0
Lsb	Lsb1-c2	0	0	26	0	180	0
	Lsb1-s1	0	0	6	0	180	0
Sb	Sb1-s1	0	0	7	0	180	0
	Sb1-j1	0	0	0	0	180	0
Pb	Pb-j1	129.037785	74.5	-5+1=-4.5	0	0	0
	Pb-j2	0	149	-5+1=-4.5	0	0	0
	Pb-j3	-129.037785	74.5	-5+1=-4.5	0	0	0
	Pb-j4	-129.037785	-74.5	-5+1=-4.5	0	0	0
	Pb-j5	0	-149	-5+1=-4.5	0	0	0
	Pb-j6	129.037785	-74.5	-5+1=-4.5	0	0	0

1) Dimensions: X, Y and Z are mm; Rx, Ry and Rz are degree.

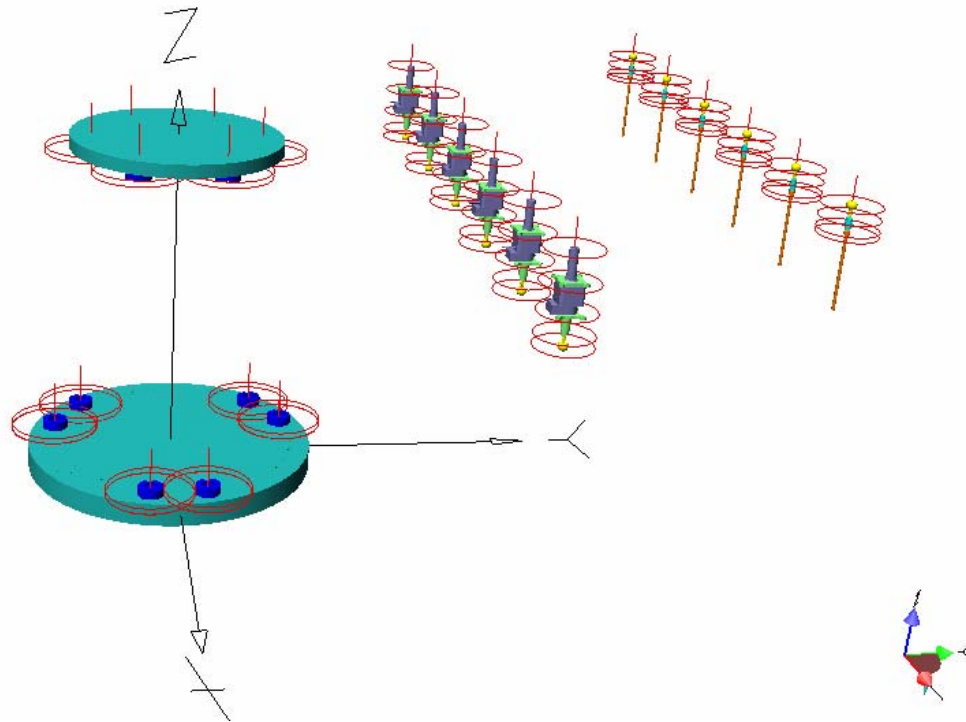


Figure 4.3 Connections of parts.

Table 4.3 Orientations of the axes for $\varphi_b = 30$ deg.

Axis	$R_x^{1)}$	$R_y^{1)}$	$R_z^{1)}$
1	-5.27128754	-2.17930829	22.3996672
2	-0.746796351	5.65389711	-82.4631226
3	4.52990963	-3.46806031	142.637208
4	-4.52990963	-3.46806031	37.3627917
5	0.746796351	5.65389711	-97.5368774
6	5.27128754	-2.17930829	157.600333

1) Dimensions: R_x , R_y and R_z are degree.

Bases of spherical joints (sa_{i-s_1} and sb_{i-s_1}) and shafts of spherical joints (lsa_{i-s_1} and lsb_{i-s_1}) are connected with spherical joint constraints. Then, rigid joint on slot constraints are defined between lma_{i-l_1} and lmb_{i-l_1} . Due to this constraint, the movements of axes will be inside of the axes orientations. After that linear motor constraints are created between lma_{i-l_1} and lmb_{i-l_1} . Where, $i = 1, 2, \dots, 6$ as numbers of axes.

Due to the upper platform is located on $(0, 0, 0.4)$ point ($z = 400$ mm), the linear motors are not closed. In order to close them, linear motors are selected. Properties

window is clicked in VisualNASTRAN. On the configuration window, point-to-point constraint option is disabled. After that, z input box is checked. Lengths of linear motors can be controlled by writing the desired value in the z input box. If $z = 0$ is written, then linear motors are fully closed. After writing desired value which describes lengths of linear motors, the point-to-point constraint option is enabled again. Thus, simulation of the hexapod is manually prepared by VisualNASTRAN.

In order to test colliding; φ_b is changed 20 deg to 30 deg. Hexapod is moved to end points and parts are observed. It is observed that no risk is occurred when $\varphi_b = 30$ deg. Therefore, the angle between closest joints on lower platform (φ_b) is decided as 30 deg.

4.3 Manufacturing of the Hexapod

After generating 2D manufacturing drawings (Appendix C), parts are manufactured. All parts are made from stainless steel (AISI304, 1.4401 standards). This section is compiled from the second TUBITAK report (Karagülle et al., 2007).


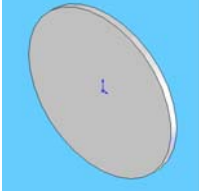
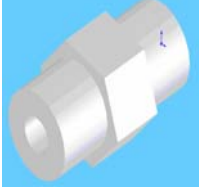

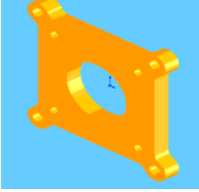

Hexapod consists of a lower platform, joints, lower connection parts, linear motors, upper connection parts, stud bolts, joint connection parts and an upper platform. Linear motors (HSI 43K4U – 05 – 032ENG) and spherical joints (Hephaist SRJ008C) are purchased. Names, numbers and perspective views of 3D solid models of manufactured parts are given in Table 4.4.

The lower platform is made by cutting with laser system (model: Trumph) and after lathing. Joint locations are precisely processed with a milling machine (model: Foreman). Platform bores of joints are located by a divider and drilled by the milling machine. The upper platform is made by the same methods applied to the lower platform.

Lower connection parts are manufactured by welding. The planar parts are cut by a laser system. Bores are marked by laser and drilled by a drilling machine.

Cylindrical parts are manufactured with CNC lathing machine (model: Goodway GLS200). Cylindrical parts are located into the centre of the planar parts and then welded. After that, the parts are cleaned by lathing. Upper connection parts are cut and marked by a laser system and drilled by the milling machine (Karagülle et al., 2007).

Table 4.4 Manufactured parts.

Part number	Part name	Items	Perspective view of the 3D solid model of parts
1	Lower platform	1	
2	Upper platform	1	
3	Joint connection part	6	
4	Lower connection part	6	
5	Upper connection part	6	
6	Stud bolt	24	

Stud bolts and joint connection parts are manufactured by CNC lathing machine. In order to prevent them from contacting to the motors, one surface of the stud bolts are milled. Joint connection parts are made by lathing cylindrical sides and milling the wrench edges.

Assembly of the hexapod is made by the following assembly flow: Firstly spherical joints are screwed to first and second parts shown in Table 4.4. Third parts (Table 4.4) are screwed to joints. Fifth parts are screwed to linear motors with 4-40 socket head cap screws (*length* = 3/8 inch). Fourth parts are attached to linear motors via fifth parts and sixth parts. This subassembly is rigidly connected by screwing fourth parts to spherical joints of the lower platform. Then shafts of linear motors are screwed to third parts which are attached to joints of the upper platform (Table 4.4). Therefore, the hexapod is manufactured and assembled (Karagülle et al., 2007).

4.4 Hexapod Control System

The hexapod control system consists of linear motor drivers, PCI motion control cards and a power supply. A schematic view of the system is shown in Figure 4.4. The control system is also shown in Figure 4.5. A portable control panel is designed and manufactured for the control system. A detailed view of the control panel is in Figure 4.6. The control panel comprises a power supply, a distributor of the power supply, PCI motion control cards, linear motor drivers and cables. In the control panel and hexapod control system, PCI 8132 card is connected to 1st and 2nd drivers, PCI 8164 card is connected to 3rd, 4th, 5th and 6th drivers. Drivers and control cards need 24VDC power for operation. This power is supplied from the 24VDC power supply and allocated by the manufactured distributor of the power supply.

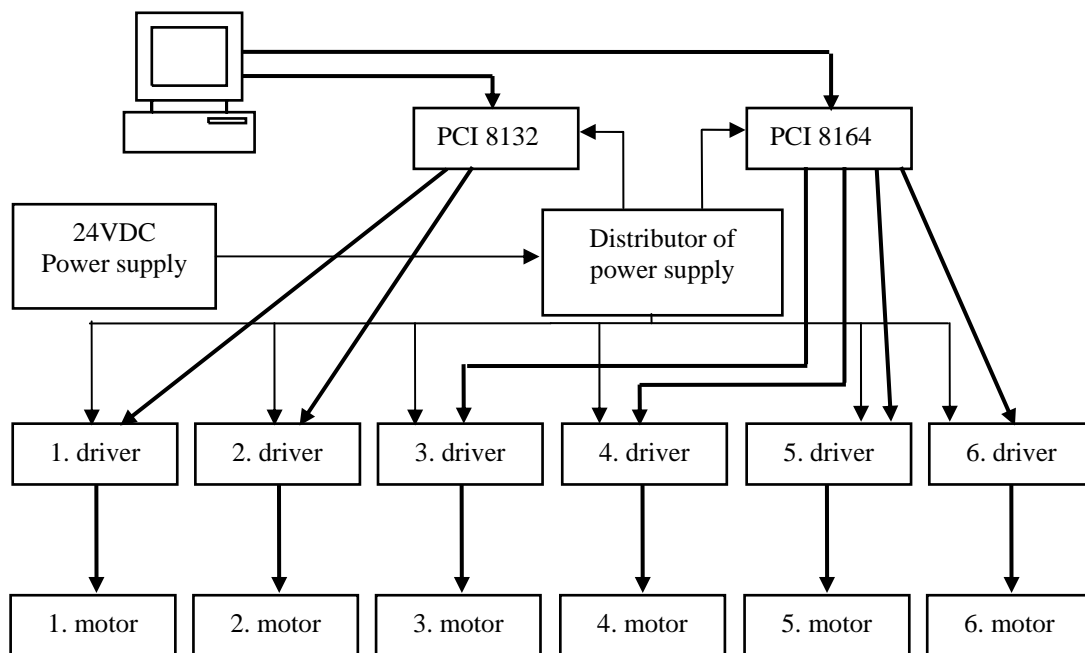


Figure 4.4 A schematic view of the control system.

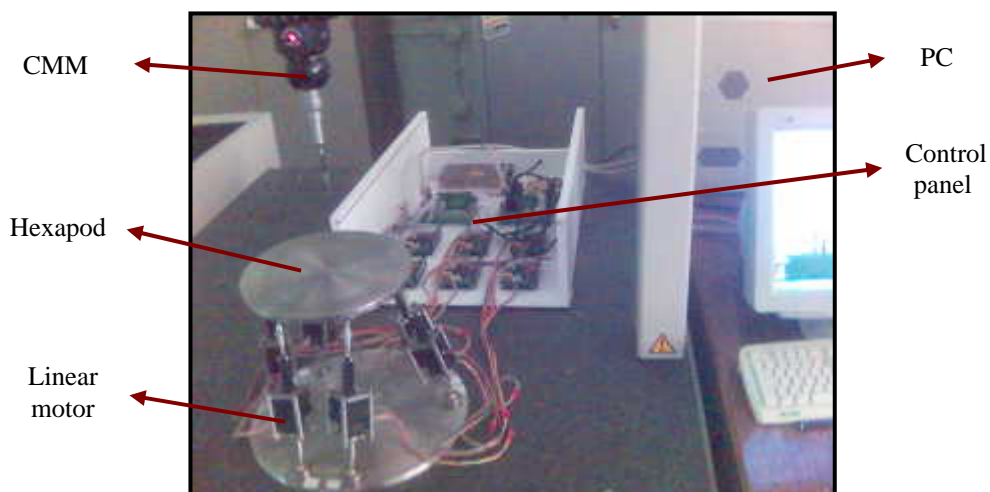


Figure 4.5 A view of the control system.

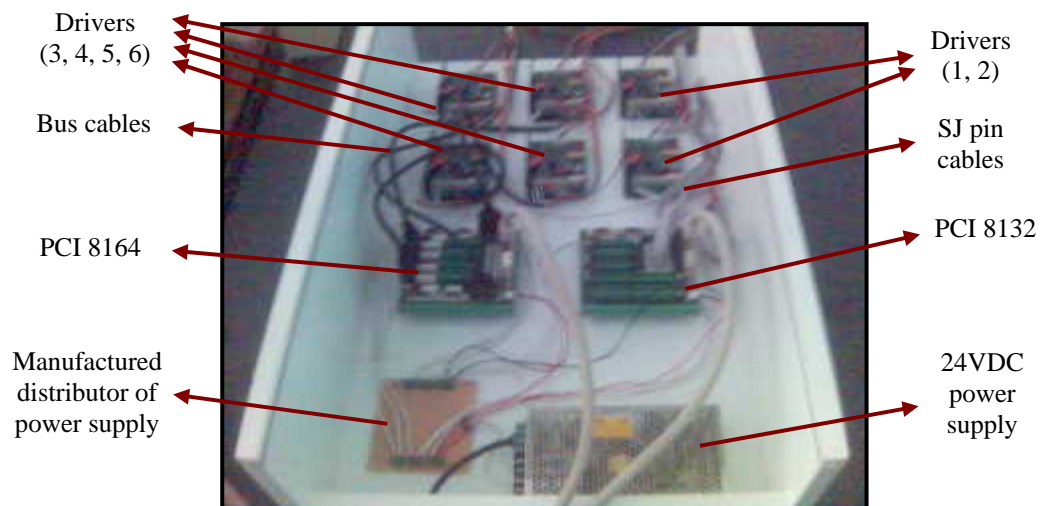


Figure 4.6 A detailed view of the control panel.

A schematic view of a linear motor driver is shown in Figure 4.7. Input ports and output ports of linear motor drivers are *Pin T2* and *Pin T1*, respectively. Connections between *Pin T2* and motion control cards make differences for PCI 8132 and PCI 8164 control cards.

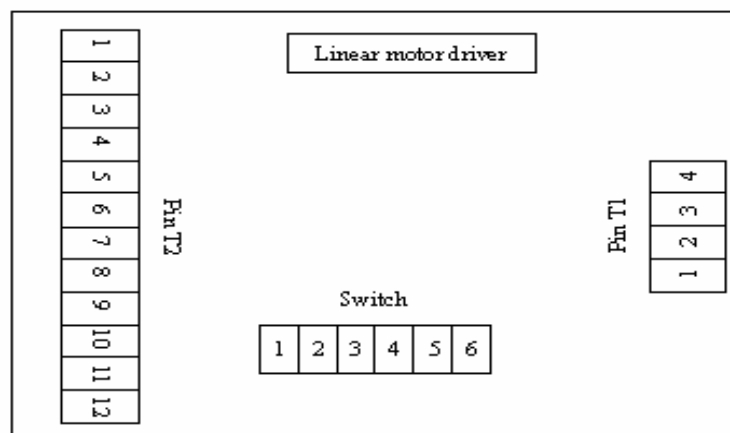


Figure 4.7 A schematic view of a linear motor driver.

Matches of required pins to drive motors, between PCI 8132 control card, power supply and *Pin T2* of linear motor drivers are presented in Table 4.5. Furthermore, matches of required pins between PCI 8164 control card, power supply and *Pin T2* of linear motor drivers are presented in Table 4.6.

Table 4.5 Pin matches between PCI 8132 control card, power supply and linear motor drivers.

Linear motor drivers (Pin T2)		SJ(1-2) pin outputs for PCI 8132	
Pin number	Signal	Pin number	Signal
3	Clock input	2	Out -
10	Direction control	4	Dir -
11	Enable control	8	Servo on
1	GND	Power supply	Ground
2	+24VIN		+24VDC

Table 4.6 Pin matches between PCI 8164 control card, power supply and linear motor drivers.

Linear motor drivers (Pin T2)		Bus cable for PCI 8164		
Pin number	Signal	Pin number	Cable shape and colour	Signal
3	Clock input	13	Pink (-)	Out -
10	Direction control	12	Red (-)	Dir -
1	GND	Power supply	Ground	
2	+24VIN		+24VDC	

1st and 2nd pins of *Pin T2* of linear motor drivers (Table 4.5 and Table 4.6) are power input pins. 3rd pin and 10th pin are for pulse (clock) input and for direction input, respectively. 11th pin is optionally used for motor enable/disable option. In order to control linear motor drivers via a PC, *J1* jumpers which are on linear motor drivers have to be opened and switches (Fig. 4.7) have to be adjusted according to Table 4.7 which gives adjustments of switches.

Table 4.7 Switch adjustments of linear motor drivers.

Switch number	1	2	3	4	5	6
Position	Open	Close	Close	Close	Close	Close

Linear motor drivers consumes 0.7 A electrical current. To set motor current: it is ensured that power is applied to motors. *Switch 3* (Table 4.7) is turned to open position. A high impedance voltmeter is connected to *VREF* + and – terminals on the

driver board. Potentiometer $P2$ on the board is adjusted until voltmeter shows the reference voltage (V_{ref}) which is as following formula:

$$V_{ref} = 0.5 \times \text{motor current} \quad (4.1)$$

Where, units of motor current and V_{ref} are ampere and volt. For the used linear motors, motor current is 0.7 A/phase. Therefore, adjusted reference voltage is 0.350 V.

Open collector type encoders have been attached to linear motors to take feedbacks. Cable connections between encoders and PCI motion control cards have to be made to take feedback signals. 1st and 4th pins of the encoder cables are for power supply. 3rd pin, 5th pin and 2nd pin of the encoder cables transmit A phase signals, B phase signals and index signals, respectively. Connecting these pins to PCI cards are different for PCI 8132 and PCI 8164. Encoder cable connections for PCI 8132 are presented in Table 4.8.

Table 4.8 Encoder cable connections for PCI 8132.

Linear motor encoder cable			SJ(3-4) pin for PCI 8132		Explanation
Cable number	Cable colour	Signal	Pin number	Signal	
1	Brown	GND	10	GND	
2	White	Index	8	EZ -	
3	Blue	Channel A	4	EA -	
5	Yellow	Channel B	6	EB -	
4	Orange	+5VDC	7	EZ +	There is +5VDC in 6 th pin of Pin T2 or in 1 st and 2 nd pins of SJ3-4. Any of these pins have to be bridged with 3 rd , 5 th and 7 th pins of bus cable and 4 th pin of encoder cable.
			3	EA +	
			5	EB +	

1st and 2nd pins of *SJ3* and *SJ4* sockets of the PCI 8132 card or 6th pins of *Pin T2* of linear motor drivers can be used to have +5VDC for encoders. *EA+*, *EB+* and *EZ+* input pins of PCI 8132 card have to be bridged with +5VDC. Because types of encoders are open collector and type of the control cards is line driver.

Encoder cable connections for PCI 8164 are presented in Table 4.9. Bus cables are connected to *CNA* sockets of the PCI 8164 card. There is no +5VDC pin in bus cables. Thus, encoder power supply is achieved by taking +5VDC from 6th pins of linear motor drivers. *EA+*, *EB+* and *EZ+* pins of the PCI 8164 card are also bridged with +5VDC similar to pins of PCI 8132.

Table 4.9 Encoder cable connections for PCI 8164.

Linear motor encoder cable			Bus cable for PCI 8164			Explanation
Cable number	Cable colour	Signal	Pin number	Cable shape and colour	Signal	
1	Brown	GND	10	White (-)	GND	
2	White	Index	15	Green (-)	EZ -	
3	Blue	Channel A	16	Blue (-)	EA -	
5	Yellow	Channel B	17	Light blue (-)	EB -	
4	Orange	+5VDC	5	Green	EZ +	There is +5VDC in 6 th pin of Pin T2. This pin has to be bridged with 5 th , 6 th and 7 th pins of the bus cable and 4 th pin of encoder cable.
			6	Blue	EA +	
			7	Light blue	EB +	

CHAPTER FIVE

SIMULATION AND CONTROLLING OF THE HEXAPOD

5.1 Introduction

In this chapter, it is desired to simulate and control the hexapod. Simulation is performed by VisualNASTRAN. Simulation gives values of inverse kinematic solution as outputs. Controlling of the hexapod is achieved by a developed VisualBASIC program. This program manages VisualNASTRAN and takes outputs of VisualNASTRAN as inputs (Karagülle et al., 2007).

The designed hexapod is shown in Figure 5.1. For this hexapod, dimensions of the fixed lower platform: diameter is 348 mm, thickness is 11 mm and the angle between closest joints is 30 deg. The centres of spherical joints are on a circle whose diameter is 298 mm and 6 mm above from the upper surface of the fixed lower platform. Dimensions of the movable upper platform: diameter is 250 mm, thickness is 10 mm and the angle between closest joints is 60 deg. The centres of spherical joints are on a circle whose diameter is 200 mm and 6 mm below from the lower surface of the movable upper platform. Distance in the z direction between the upper surface of the lower platform and the lower surface of the upper platform is 233.1362 mm when the actuators, consequently the hexapod, are fully closed. Stroke of the linear motors is 50.8 mm and maximum tilt angle of the joints is 45 deg. Global z axis is perpendicular to the line which ties centres of the 1st spherical joint and the 6th spherical joint. Global x - y plane is parallel to the surface of the lower platform. Direction of the global z axis is along a line which goes to the upper platform from the lower platform.

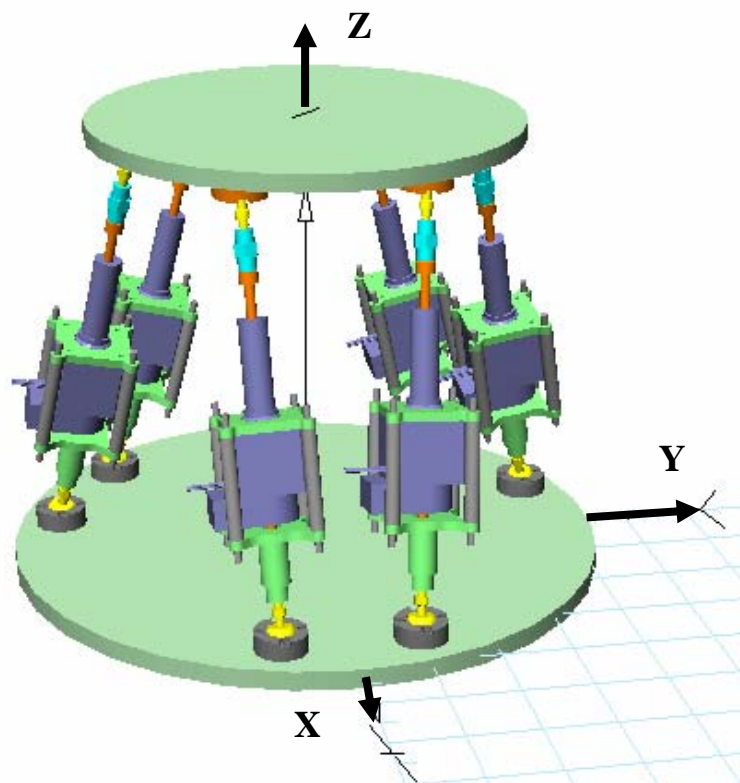


Figure 5.1 Designed hexapod

5.2 Simulation and Controlling of the Hexapod

5.2.1 A Developed VisualBASIC Program for Simulation and Control

VisualNASTRAN has API (Application Programming Interface) property. Therefore, it can be programmed by VisualBASIC. By programming VisualNASTRAN, design and inputs can be parametrical. Any changes are possible by changing the parameters. Application can be easy and rapid. Simulation and controlling of hexapod are achieved by a VisualBASIC program. This program assembles solid parts, simulates the hexapod and solves inverse kinematic analyses and controls linear motors as point-to-point application via PCI motion control cards. The name of the program is “hexapod1.vbp”. It contains “hexapod1.frm” and modules listed in Table 5.1 (Karagülle et al., 2007).

Table 5.1 Modules of the VisualBASIC project named hexapod1.vbp.

assemble.bas	Assembles hexapod in VisualNASTRAN
simulate.bas	Makes inverse kinematic analysis of hexapod and determines length of linear actuators.
move.bas	Determines parameters of ADLINK PCI motion control cards by using lengths, which come from simulation, of linear actuators and moves hexapod to desired position.
adlink_init.bas	Initials ADLINK PCI motion control cards.
pci_8132.bas	Contains commands of ADLINK PCI 8132 motion control card. This module comes with the installation CD of PCI motion control cards.
pci_8164.bas	Contains commands of ADLINK PCI 8164 motion control card. This module comes with the installation CD of PCI motion control cards.

First of all, components of PCI motion control cards have to be added onto the form. They can be added to project by selecting from menu: *project > components window* and “DAQBench PCI8132 Motion Control Card ActiveX Control” and “DAQBench PCI8164 Motion Control Card ActiveX Control”. These components have to be inserted onto the form. Then, the names are changed as B_8132 for PCI 8132 and B_8164 for PCI 8164. After that, Pci_8132.bas and Pci_8164.bas modules are inserted into the project. Thus, preparations of control cards are finished.

When the developed VisualBASIC program is run, Figure 5.2 is viewed. The program is given in Appendix D (Karagülle et al., 2007).

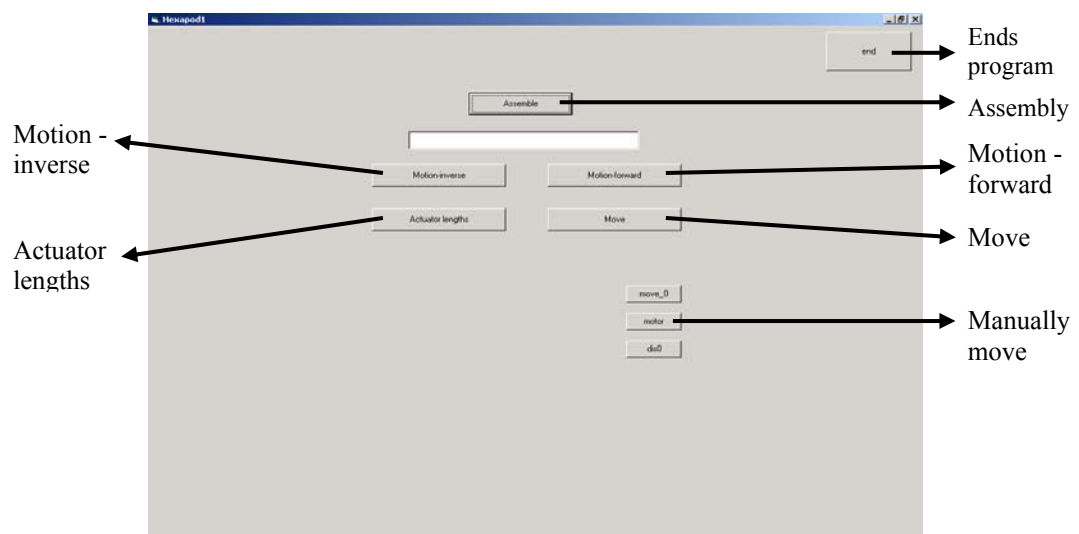


Figure 5.2 Interface of the program.

5.2.2 Simulation and Inverse Kinematic Analysis of the Hexapod

A VisualNASTRAN model is constructed with simple 3D models according to real hexapod, in order to have a rapid process. Detailed 3D models might be slowed the programs. There are an upper moving platform (pb), a fixed lower platform (pa), a rectangular part (lsa) which represents constant part of a link and a cylindrical part (lsb) which represents moving part of a link in the basic kinematic model. These parts are in dimensions of designed hexapod. These parts can be seen in Figure 5.3 (Karagülle et al., 2007).

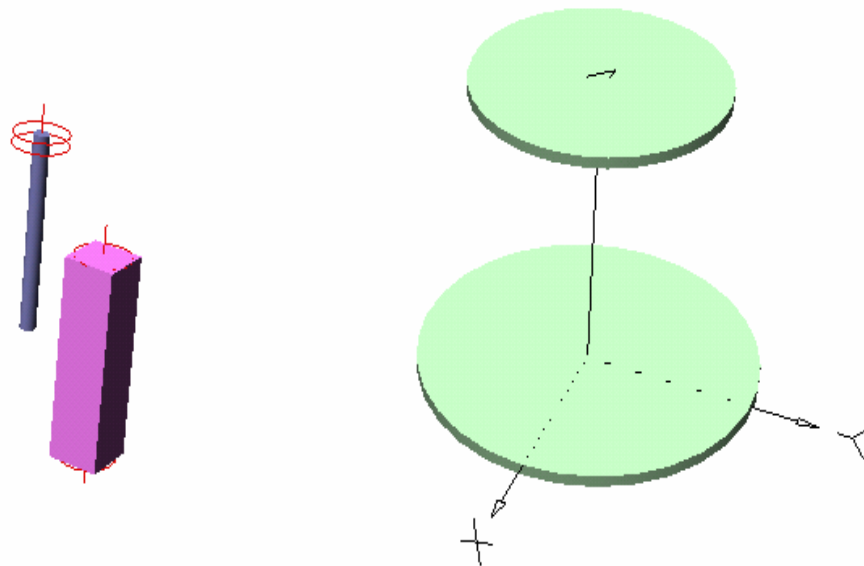


Figure 5.3 Basic model parts of the hexapod

Two axes are added to the lsa and lsb parts for assembly. Then five copies are created. The file is saved as “hexapod1.wm3”. The VisualBASIC program which is described above is run. By clicking “assembly” command button, parts are automatically renamed, aligned to their attachment directions. Constraints such as rigid joints, spherical joints and linear motors are created. In order to define “prescribed motion” to upper platform, sliders are inserted for V_x , V_y , V_z , W_x , W_y and W_z of upper platform. These sliders are used as input boxes. To measure length changes of links, meters are assigned to linear motors. Force values of linear motors

are set as zero. Also, in order to determine the position and orientation of the upper platform, meters are assigned to it. Thus, assembly is finished. The file is saved as “hexapodi.wm3”. The model can be seen in Figure 5.4 (Karagülle et al., 2007).

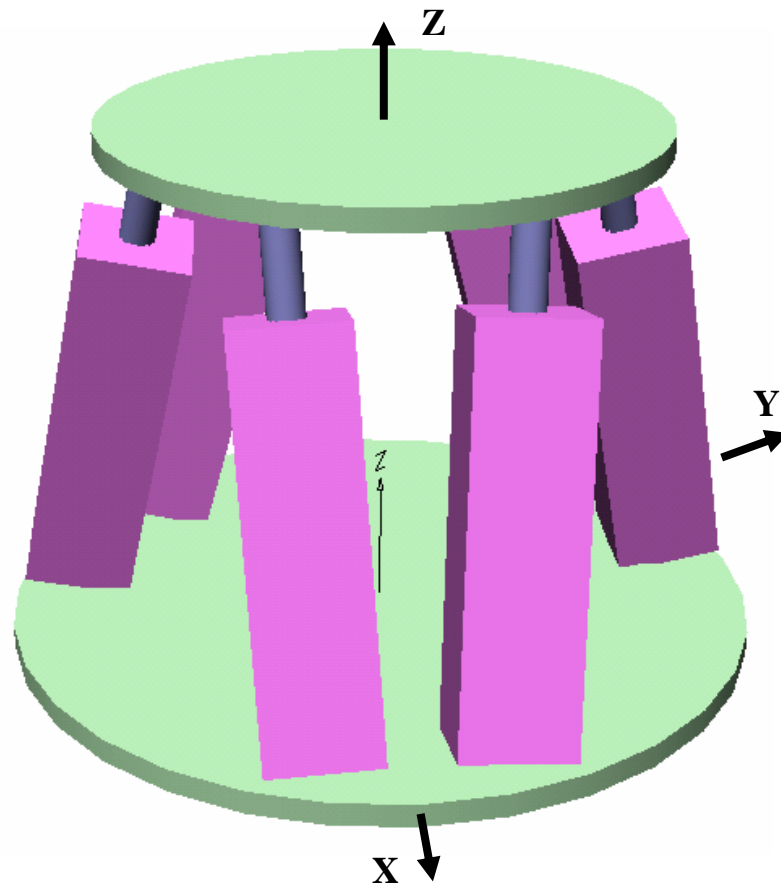


Figure 5.4 Basic model of the hexapod.

Input values of x_p , y_p , z_p , θ_{xp} , θ_{yp} and θ_{zp} are inserted into `inp_inverse.txt` and the file is saved. x_p , y_p and z_p are desired positions of the movable upper platform in mm along x , y and z directions, respectively. When all linear motors are fully retracted, $x_p = 0$, $y_p = 0$ and $z_p = 0$. Assume that, movable axis system which is in the centre of upper platform is x_l , y_l and z_l . Desired rotation about x_l is θ_{xp} ; desired rotation about y_l is θ_{yp} and then desired rotation about z_l is θ_{zp} . Angles are in degrees. When all linear motors are fully retracted, $\theta_{xp} = 0$, $\theta_{yp} = 0$ and $\theta_{zp} = 0$.

When “motion inverse” command button is clicked, the VisualBASIC program takes actual position information of hexapod from VisualNASTRAN program in a

loop. Error is generated by subtracting actual position from desired position. The VisualBASIC program multiplies error with gain coefficient and sends velocity values to VisualNASTRAN, and it is run for one step. The loop is finished when the upper platform reaches desired position with acceptable errors. Simulation is done and length changes of linear motors are generated by meters which are attached to linear motors (Karagülle et al., 2007).

After simulation is done, “actuator lengths” command button is clicked. Therefore, length outputs of linear motors can be taken from VisualNASTRAN. Tilt angles of spherical joints are computed with these outputs. Then a comparison about lengths of linear motors and tilt angles is started in order to determine whether the hexapod is in workspace or not. If angles or lengths exceed their maximum values, the hexapod is not in the workspace and then the program gives an error message (Karagülle et al., 2007).

5.2.3 Controlling of the Hexapod

Open loop position control is applied to hexapod via linear motors. Point – to – point applications are processed. PC-based motion control is achieved over ADLINK PCI 8132 and PCI 8164 motion control cards.

Control of hexapod manipulator is based on inverse kinematics (Jelenkovic et al., 2004). The simulation (inverse kinematic solution) gives length changes of linear motors with respect to time. These values are taken as inputs. If the “Move” command button is clicked on the VisualBASIC interface (see Fig. 5.2), pulse numbers which will be applied to linear motors are determined with respect to the difference between initial position and final position of the moving upper platform.

In order to start and stop movement simultaneously, velocities are determined by using interpolation. The maximum velocity is chosen as 3000 pulse/s. The biggest distance extracted (or retracted) linear motor has the biggest velocity (3000 pulse/s). The slowest linear motor with respect to the motion moves the less.

Hexapod which has been manufactured is moved by simulation results after determining command parameters of motion control cards with respect to position and velocity data. Fundamental commands of the motion control cards are given in Appendix A.

T velocity profile command is used fundamentally in the program. The commands are as following formulas (5.1) and (5.2):

$$B_8164.StartTAMove(axis, pos, svel, mvel, tacc, tdec) \quad (5.1)$$

$$B_8132.StartTMove(axis, pos, svel, mvel, tacc, tdec) \quad (5.2)$$

Where, *axis* is the working axis number, *pos* is position input (pulse), *svel* is starting velocity (pulse/s), *mvel* is maximum velocity (pulse/s), *tacc* is acceleration time (s) and *tdec* is deceleration time (s). For the applications, they are taken that *svel* = 0 and *tacc* = *tdec* = 0.01 s. *mvel* is determined as 3000 pulse/s for the axis which has maximum length change. *mvel* for other axes is calculated by interpolation explained above.

Pulse numbers are position information. Pulse numbers are calculated according to lengths of axes (linear motors), as follows:

$$pos_i = \frac{dis_i}{0.00075}, \quad i = 1, 2, \dots, 6 \quad (5.3)$$

Where, *dis* is length changes, which is found by inverse kinematic solution, of the linear motors, *pos* is numbers of pulses. If 1 pulse is sent, then linear motors move 0.75 μm ; because step motors are bipolar. Precision of motors is 1.5 μm , 2 pulses give this value regarding bipolar type. The entire control codes are in Appendix D (Karagülle et al., 2007). Consequently, commands (5.1) and (5.2) are sent linear motors as point – to – point open loop control via motion control cards.

CHAPTER SIX

TEST RESULTS OF THE HEXAPOD

6.1 Introduction

In this chapter, test results of the hexapod whose manufacturing is discussed in Chapter 4 are presented. Tests are made by the developed VisualBASIC program which is explained in Chapter 5. Test results occurred from desired position (x_p, y_p, z_p) and orientation $(\theta_{xp}, \theta_{yp}, \theta_{zp})$, which are explained in Chapter 5, of the movable upper platform are measured by a coordinate measuring machine (CMM). Model of the CMM is Euro-C-A9106 of Mitutoyo Company (Mitutoyo Corp., 2006). Precision of the CMM is 5 μm .

6.2 Tests and Results

Positions of the upper platform are measured by CMM. In order to test, global axis system of the CMM is translated to global axis system of the hexapod. x_p and y_p is measured by creating a circle element using lateral faces of the upper platform. z_p , α_p , β_p and γ_p is determined by measuring the plane of upper surface of the upper platform. α_p , β_p and γ_p are angles between x , y and z axes and z_l axis which is attached to upper platform, respectively. α_p , β_p and γ_p can be generated from θ_{xp} , θ_{yp} and θ_{zp} with a transformation. The transformation codes is in Sub CMM () subroutine of the developed VisualBASIC program which is presented in Appendix D.

At the initial position of the upper platform, position and rotation are $x_p = 0$, $y_p = 0$, $z_p = 0$, $\alpha_p = 90$, $\beta_p = 90$ and $\gamma_p = 0$ and length changes of linear motors are zero. The upper platform is moved 14 times to different positions and then returned to the initial position. Average values are found for initial position errors. Average values are: $x_{p0} = 189.6429 \mu\text{m}$, $y_{p0} = -179.7143 \mu\text{m}$, $z_{p0} = -36.7 \mu\text{m}$; $\alpha_0 = 89.97167 \text{ deg}$, $\beta_0 = 90.03476 \text{ deg}$, and $\gamma_0 = 0.04667 \text{ deg}$. Therefore, errors at the initial position for x_p , y_p , z_p , α_p , β_p and γ_p are calculated by subtracting x_p , y_p , z_p , α_p , β_p , and γ_p from x_{p0} , y_{p0} , z_{p0} , α_0 , β_0 , and γ_0 , respectively. Thus, errors are $\epsilon_{x_0} = 189.6429 \mu\text{m}$, $\epsilon_{y_0} = -179.7143 \mu\text{m}$,

$\varepsilon_{z0} = -36.7 \mu\text{m}$, $\varepsilon_{\alpha0} = -0.02833 \text{ deg}$, $\varepsilon_{\beta0} = 0.03476 \text{ deg}$, $\varepsilon_{\gamma0} = 0.04667 \text{ deg}$, respectively. Test results for initial position errors can be shown in Figure 6.1. Initial position errors and maximum deviations from averages can be found in Table 6.1. Maximum deviations from averages give repeatability.

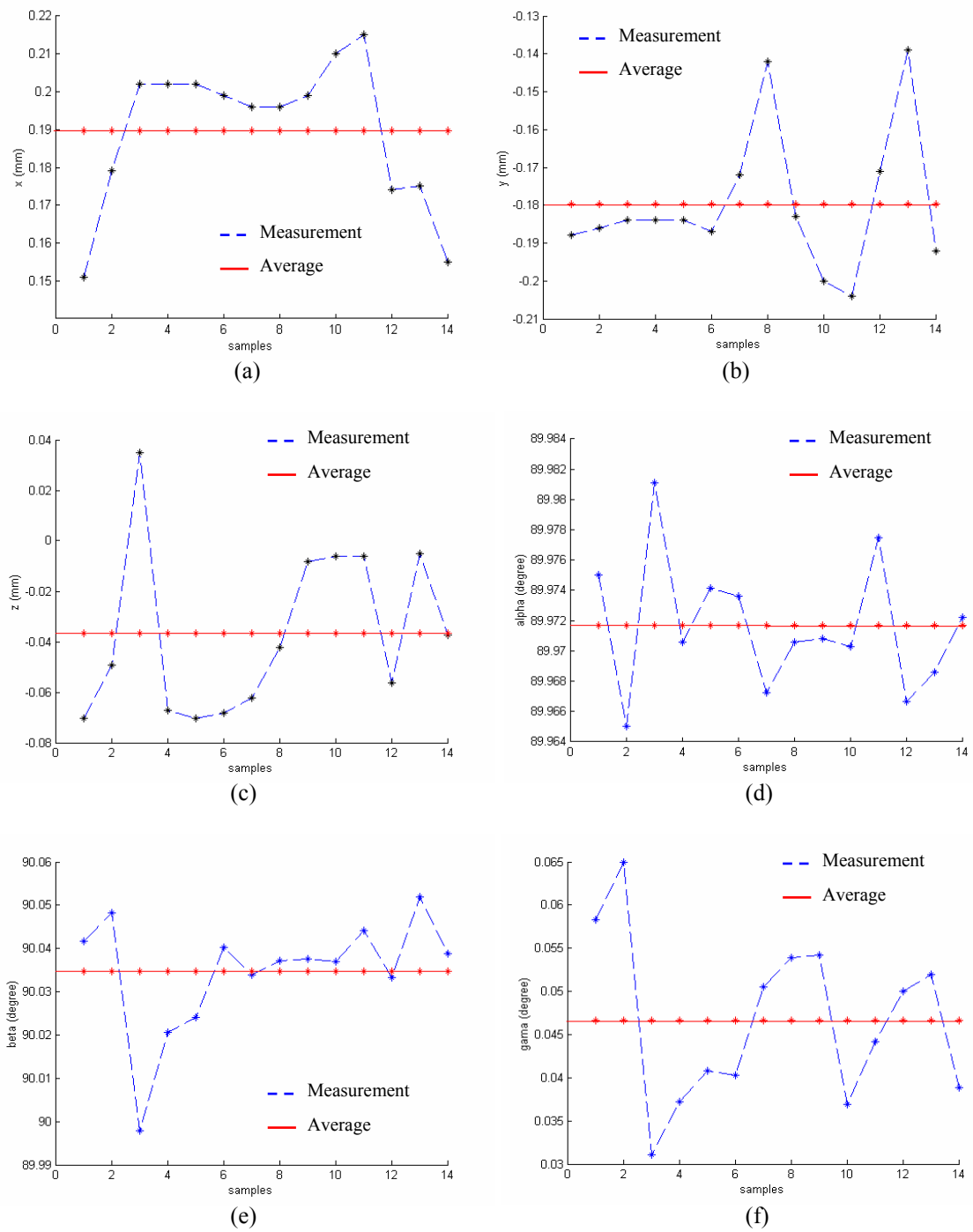


Figure 6.1 (a) x_p , (b) y_p , (c) z_p , (d) α_p , (e) β_p and (f) γ_p measurement results of initial position tests.

Table 6.1 Results of initial position.

Desired position ¹ x_p, y_p, z_p $\alpha_p, \beta_p, \gamma_p$	Averages of errors ^{2,3} $\epsilon_x, \epsilon_y, \epsilon_z$ $\epsilon_\alpha, \epsilon_\beta, \epsilon_\gamma$	Maximum deviations from averages, respectively ^{2,3}
0, 0, 0 90, 90, 0	189.6429, -179.7143, -36.7 -0.02833, 0.03476, 0.04667	38.6, 40.7, 71.5 0.00944, 0.03698, 0.01833

- 1) Units of displacements and angles are mm and degree, respectively.
- 2) Units of displacements and angles are μm and degree, respectively.
- 3) Error = measured position – desired position.

The upper platform is moved to three different positions from the initial position for 10 times with $\alpha_p = 90$ deg, $\beta_p = 90$ deg, $\gamma_p = 0$ deg. The upper platform is manually adjusted for initial position after every movement. Test results are measured by CMM. Test results of $x_p = 5, y_p = 0, z_p = 10, \alpha_p = 90, \beta_p = 90$ and $\gamma_p = 0$ position; $x_p = -2.5, y_p = 4.33013, z_p = 10, \alpha_p = 90, \beta_p = 90$ and $\gamma_p = 0$ position and $x_p = -2.5, y_p = -4.33013, z_p = 10, \alpha_p = 90, \beta_p = 90$ and $\gamma_p = 0$ position are presented in Figure 6.2, Figure 6.3 and Figure 6.4, respectively. Desired positions, averages of errors obtained from measured positions and maximum deviations from averages at initial position are given in Table 6.2.

Table 6.2 Results of (5, 0, 10; 90, 90, 0), (-2.5, 4.33013, 10; 90, 90, 0) and (-2.5, -4.33013, 10; 90, 90, 0) positions.

Desired position ¹ x_p, y_p, z_p $\alpha_p, \beta_p, \gamma_p$	Averages of errors ^{2,3} $\epsilon_x, \epsilon_y, \epsilon_z$ $\epsilon_\alpha, \epsilon_\beta, \epsilon_\gamma$	Maximum deviations from averages, respectively ^{2,3}
5, 0, 10 90, 90, 0	20.3429, 21.6857, -43.4999 -0.0090, 0.0274, 0.0243	66.7, 64.4, 10 0.0235, 0.0138, 0.0268
-2.5, 4.33013, 10 90, 90, 0	184.9429, -87.7843, -36.1999 0.00003, 0.0098, 0.0084	63.3, 81.8, 31.7 0.0106, 0.0239, 0.0106
-2.5, -4.33013, 10 90, 90, 0	174.6429, 117.5557, -44.1 0.0131, 0.0295, 0.0081	64, 41.4, 8.6 0.0117, 0.0061, 0.0183

- 1) Units of displacements and angles are mm and degree, respectively.
- 2) Units of displacements and angles are μm and degree, respectively.
- 3) Error = desired position – measured position + initial position errors.

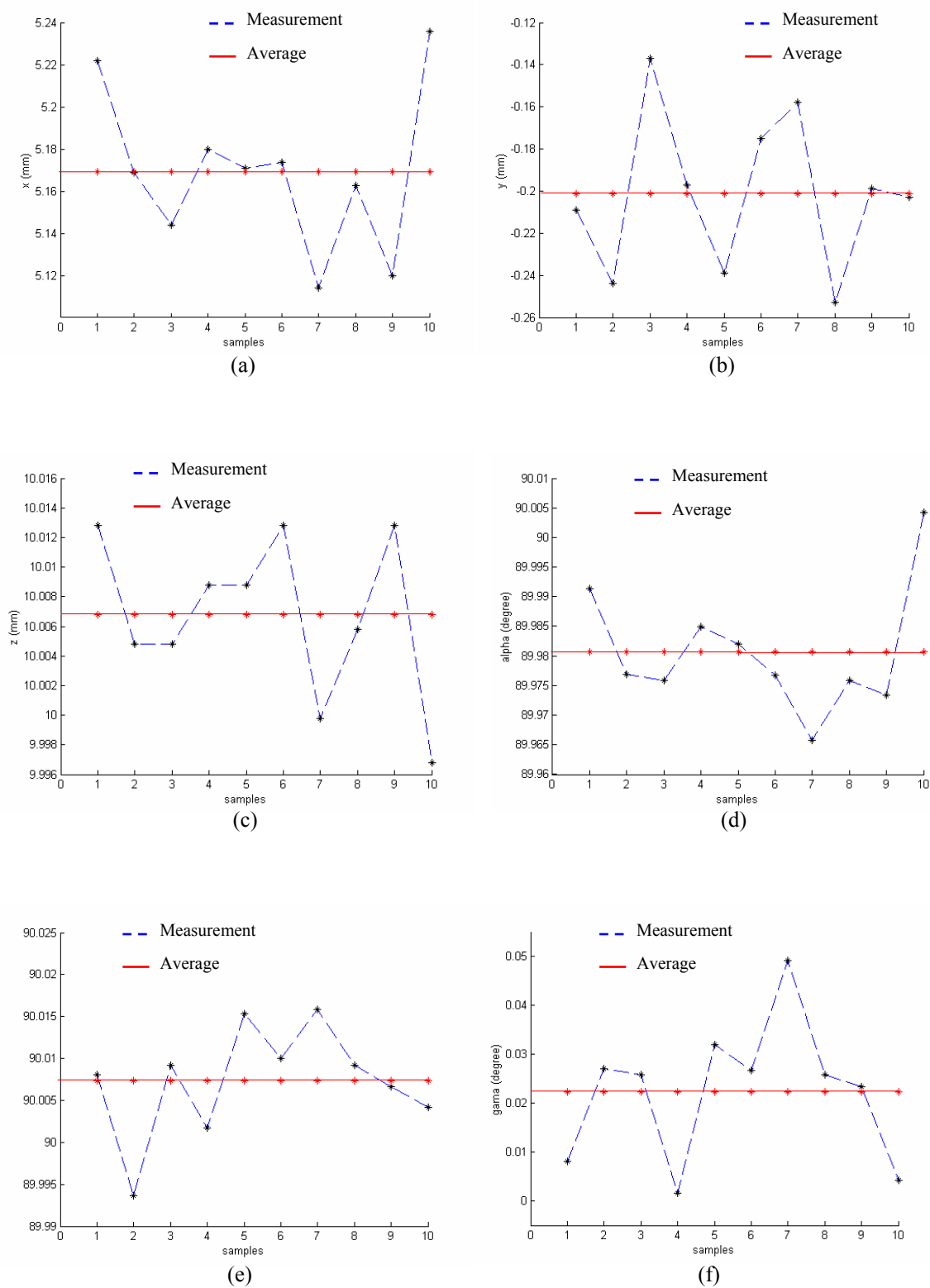


Figure 6.2 (a) x_p , (b) y_p , (c) z_p , (d) α_p , (e) β_p and (f) γ_p measurement results for desired position ($x_p = 5, y_p = 0, z_p = 10, \alpha_p = 90, \beta_p = 90$ and $\gamma_p = 0$).

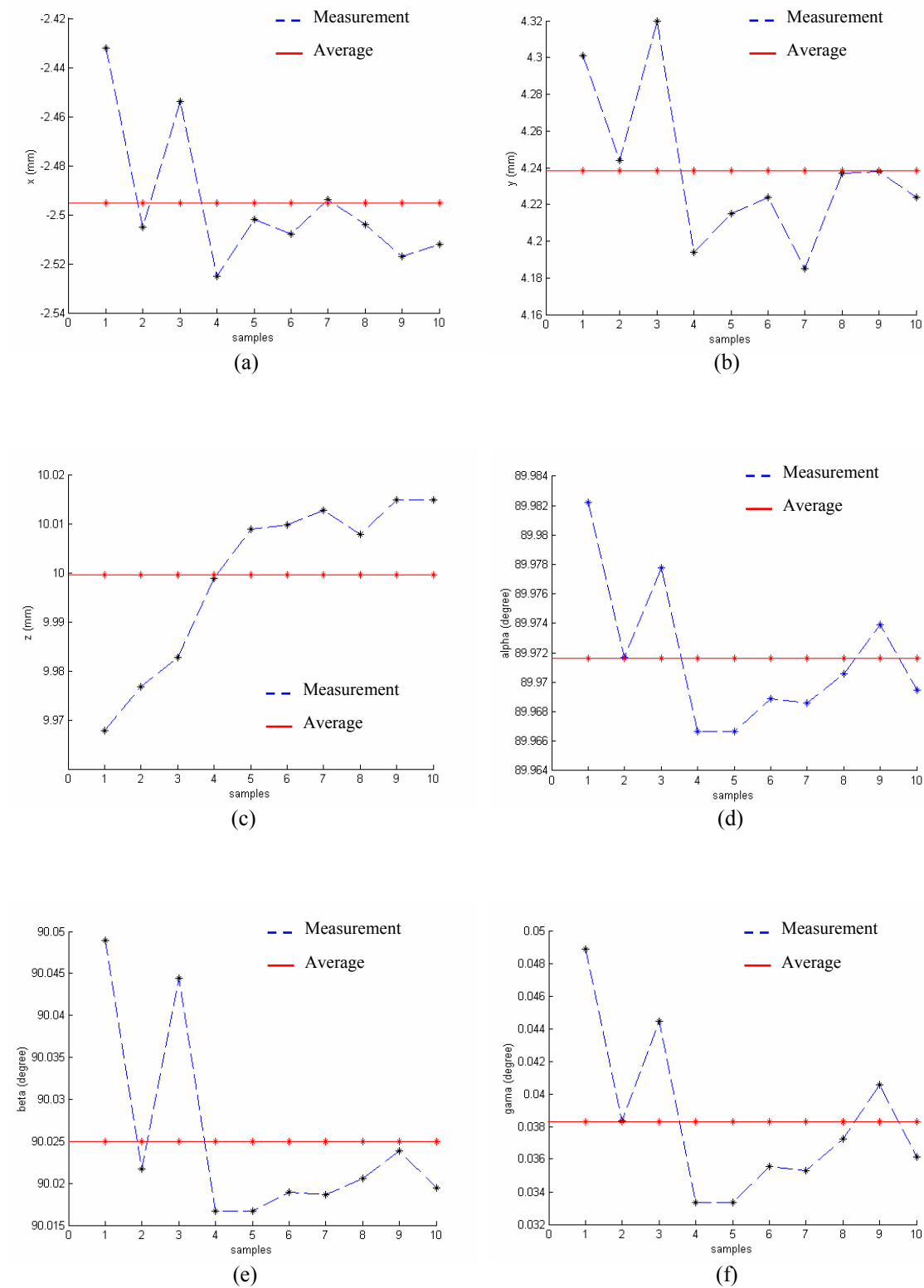


Figure 6.3 (a) x_p , (b) y_p , (c) z_p , (d) α_p , (e) β_p and (f) γ_p measurement results for desired position ($x_p = -2.5$, $y_p = 4.33013$, $z_p = 10$, $\alpha_p = 90$, $\beta_p = 90$ and $\gamma_p = 0$).

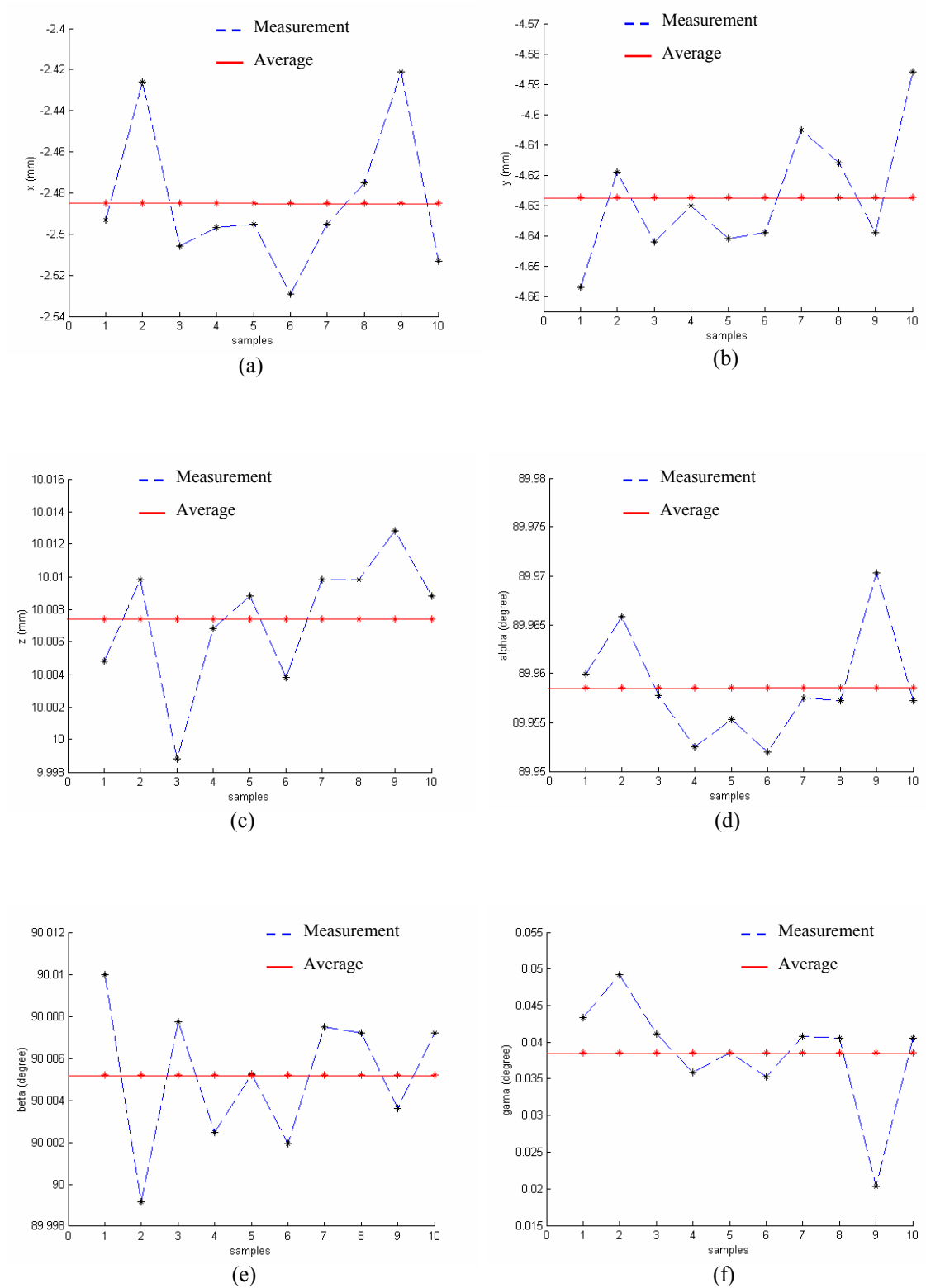
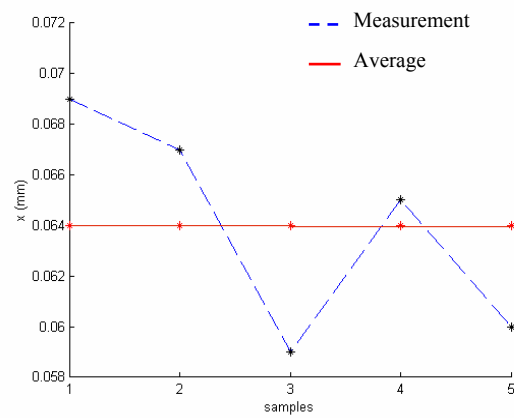
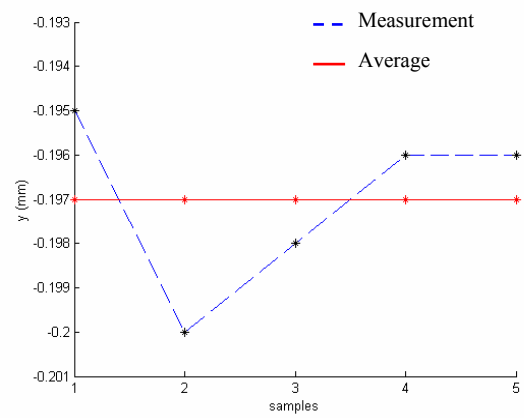


Figure 6.4 (a) x_p , (b) y_p , (c) z_p , (d) α_p , (e) β_p and (f) γ_p measurement results for desired position ($x_p = -2.5$, $y_p = -4.33013$, $z_p = 10$, $\alpha_p = 90$, $\beta_p = 90$ and $\gamma_p = 0$).

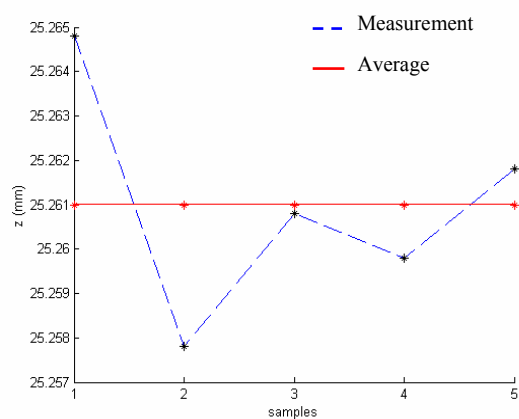
The upper platform is moved to (0, 0, 25; 90, 90, 0) position from (0, 0, 0; 90, 90, 0) position. Then, the upper platform is moved from (0, 0, 25; 90, 90, 0) position to (5, 5, 35; 90, 90, 0) position; then, from this position to (-5,-5, 15; 90, 90, 0) position; then, from previous position to (5, -5, 35; 90, 90, 0) position and then from last position to (-5, 5, 15; 90, 90, 0) position for 5 times. Test results are measured by CMM. Test results of translational motions when the upper platform is parallel ($\alpha_p = 90$, $\beta_p = 90$ and $\gamma_p = 0$) to the lower platform are presented in Figure 6.5, Figure 6.6, Figure 6.7, Figure 6.8, Figure 6.9, and Table 6.3. The table contains desired positions, averages of errors obtained from measured positions and maximum deviations from averages.



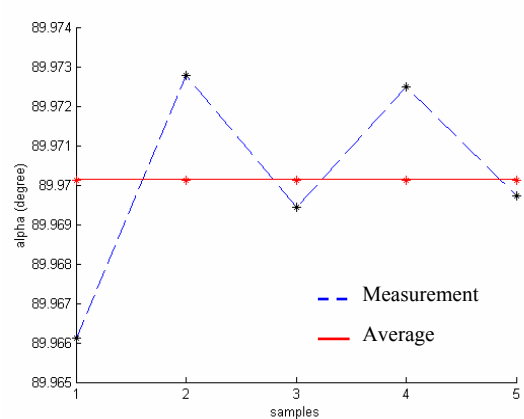
(a)



(b)



(c)



(d)

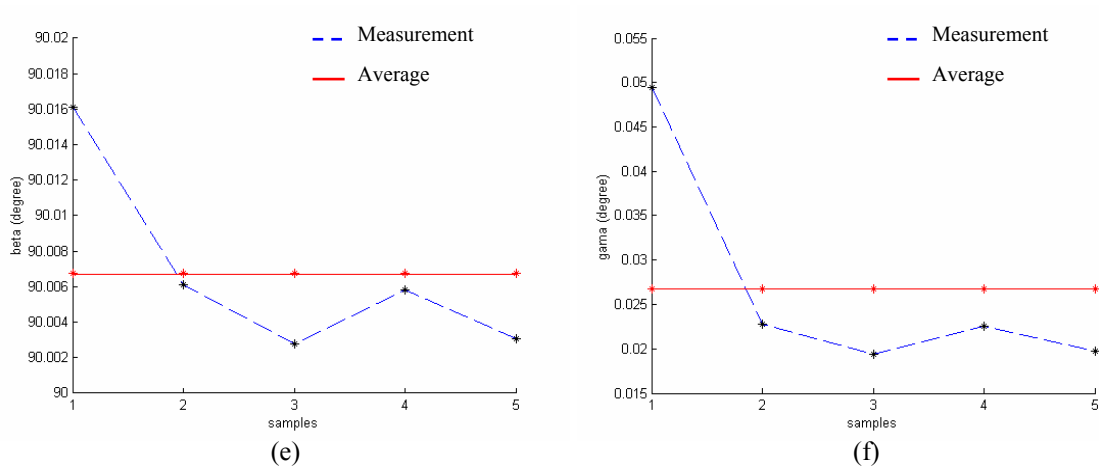
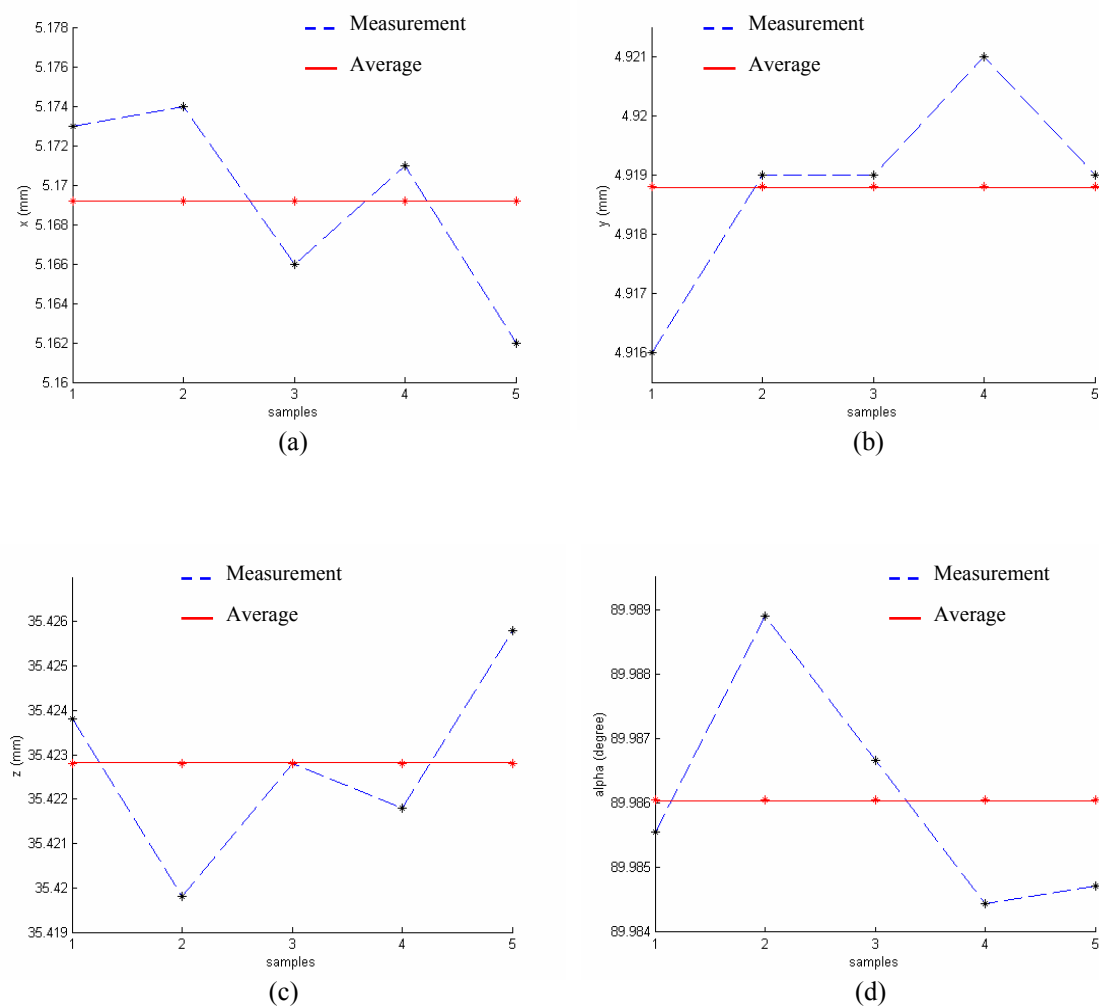
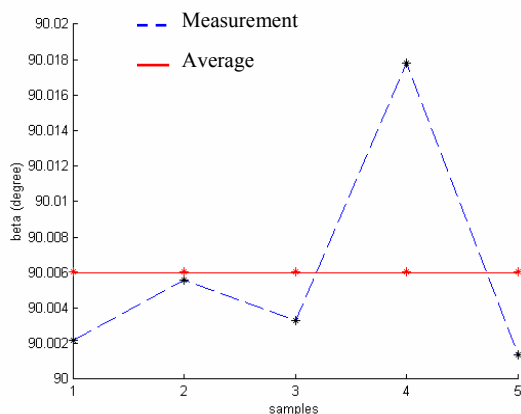
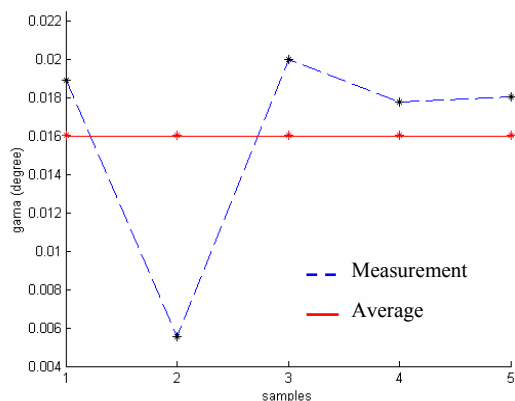


Figure 6.5 (a) x_p , (b) y_p , (c) z_p , (d) α_p , (e) β_p and (f) γ_p measurement results for desired position ($x_p = 0$, $y_p = 0$, $z_p = 25$, $\alpha_p = 90$, $\beta_p = 90$ and $\gamma_p = 0$).



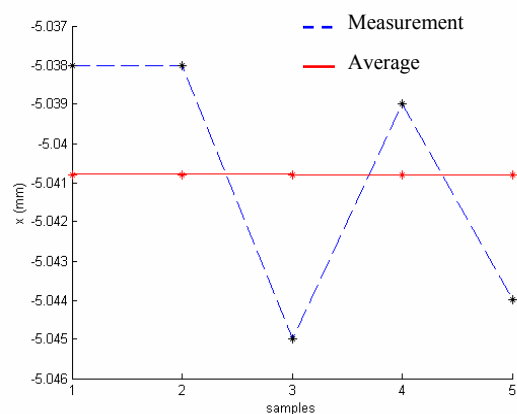


(e)

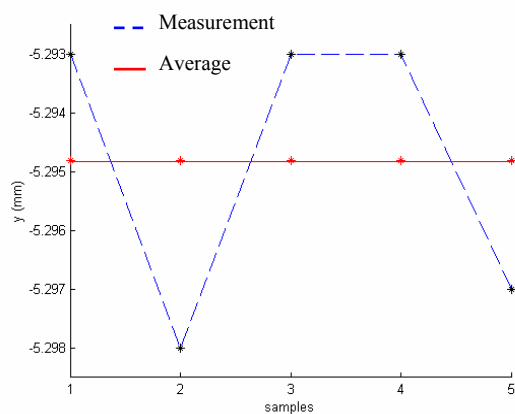


(f)

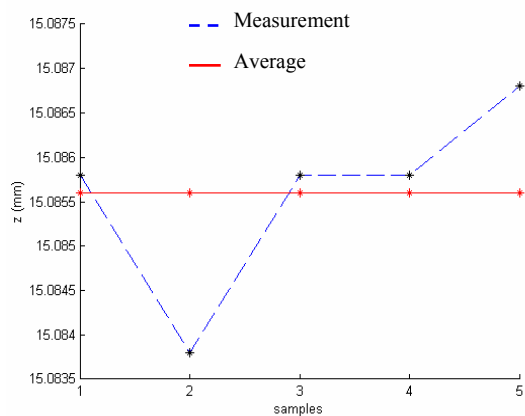
Figure 6.6 (a) x_p , (b) y_p , (c) z_p , (d) α_p , (e) β_p and (f) γ_p measurement results for desired position ($x_p = 5$, $y_p = 5$, $z_p = 35$, $\alpha_p = 90$, $\beta_p = 90$ and $\gamma_p = 0$).



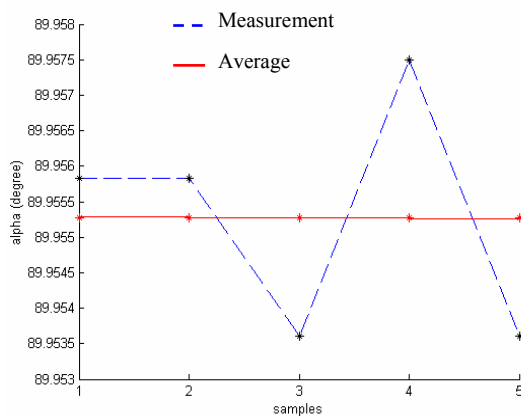
(a)



(b)



(c)



(d)

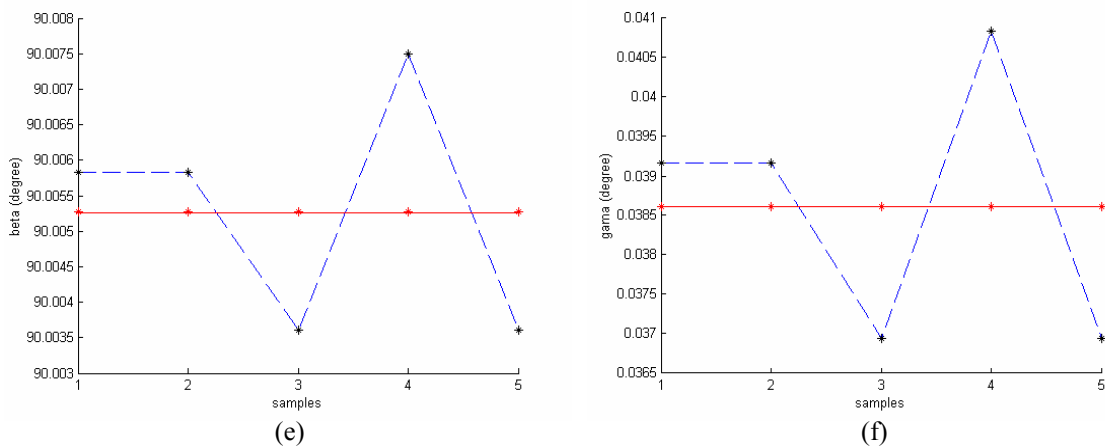
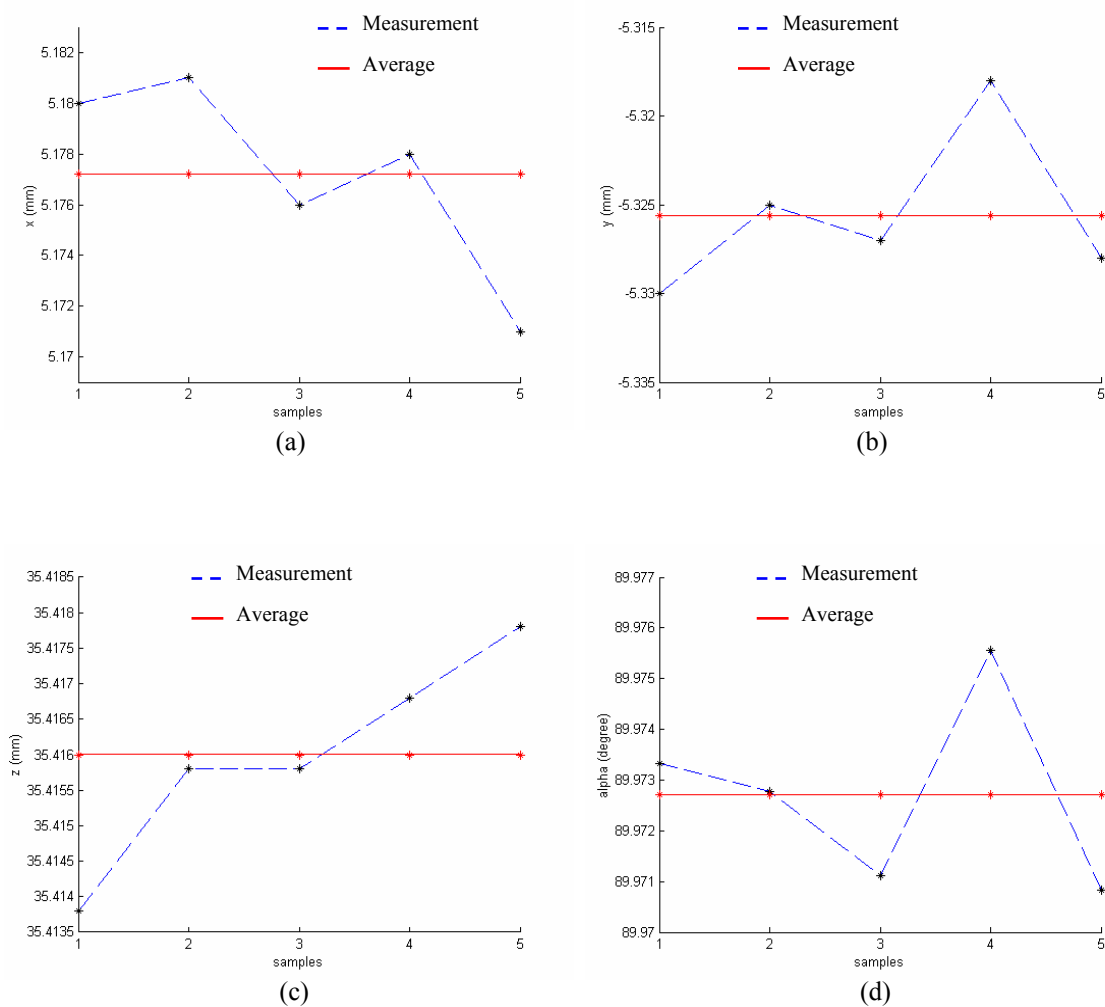


Figure 6.7 (a) x_p , (b) y_p , (c) z_p , (d) α_p , (e) β_p and (f) γ_p measurement results for desired position ($x_p = -5$, $y_p = -5$, $z_p = 15$, $\alpha_p = 90$, $\beta_p = 90$ and $\gamma_p = 0$).



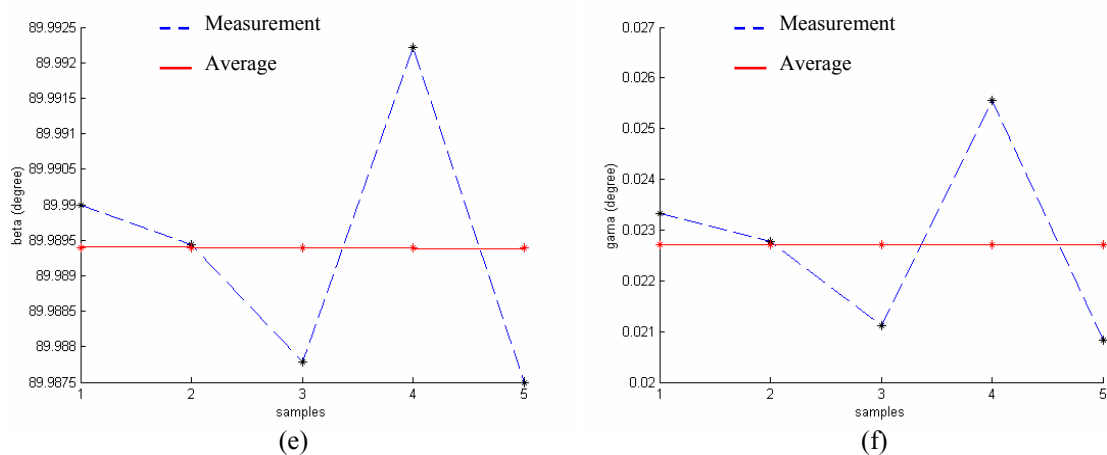
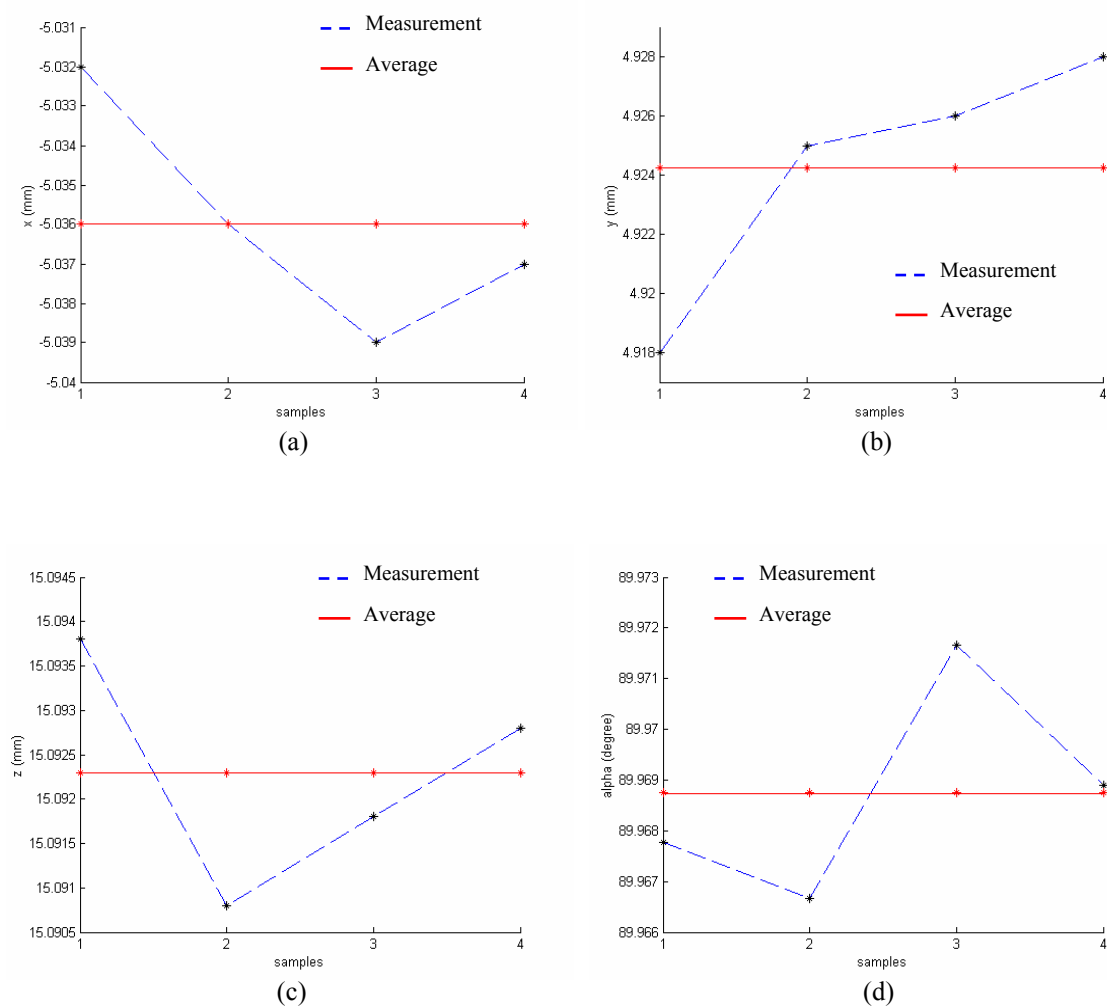


Figure 6.8 (a) x_p , (b) y_p , (c) z_p , (d) α_p , (e) β_p and (f) γ_p measurement results for desired position ($x_p = 5$, $y_p = -5$, $z_p = 35$, $\alpha_p = 90$, $\beta_p = 90$ and $\gamma_p = 0$).



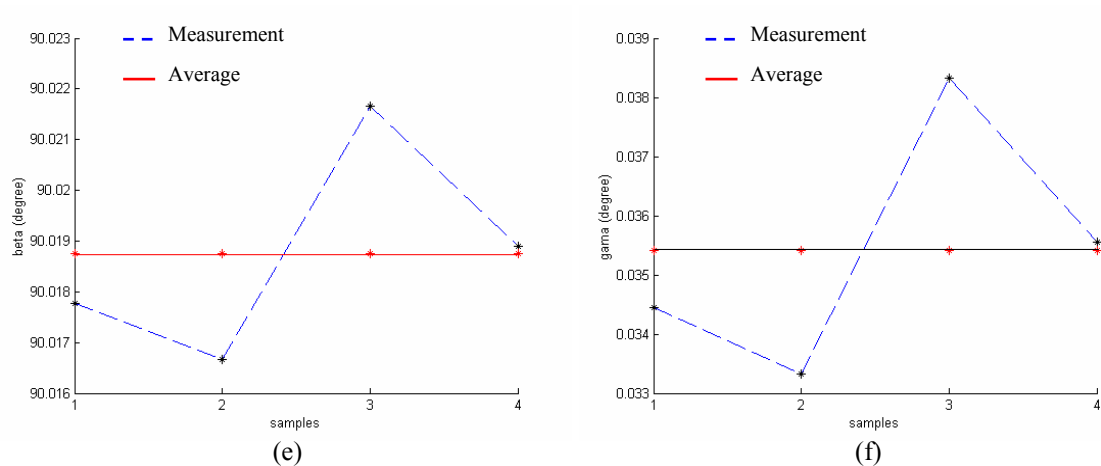


Figure 6.9 (a) x_p , (b) y_p , (c) z_p , (d) α_p , (e) β_p and (f) γ_p measurement results for desired position ($x_p = -5$, $y_p = 5$, $z_p = 15$, $\alpha_p = 90$, $\beta_p = 90$ and $\gamma_p = 0$).

Table 6.3 Results of specified positions.

Desired position ¹ x_p, y_p, z_p $\alpha_p, \beta_p, \gamma_p$	Averages of errors ^{2,3} $\epsilon_{x_p}, \epsilon_{y_p}, \epsilon_{z_p}$ $\epsilon_{\alpha_p}, \epsilon_{\beta_p}, \epsilon_{\gamma_p}$	Maximum deviations from averages, respectively ^{2,3}
0, 0, 25 90, 90, 0	125.6429, 17.2857, -297.7 0.00156, 0.02798, 0.01989	5, 3, 3.8 0.0040, 0.0093, 0.0227
5, 5, 35 90, 90, 0	20.4429, -98.5143, -459.5 -0.01439, 0.02870, 0.03061	7.2, 2.8, 3 0.00283, 0.01172, 0.0105
-5, -5, 15 90, 90, 0	230.4429, 115.0857, -122.3 0.01639, 0.02948, 0.00806	4.2, 3.2, 1.8 0.00222, 0.00222, 0.00222
5, -5, 35 90, 90, 0	12.4429, 145.8857, -452.7 -0.00105, 0.04537, 0.02395	6.2, 7.6, 2.2 0.00283, 0.00283, 0.00283
-5, 5, 15 90, 90, 0	225.6429, -103.9643, -128.99 0.00292, 0.01601, 0.01125	4, 6.2, 1.5 0.00292, 0.00292, 0.00292

- 1) Units of displacements and angles are mm and degree, respectively.
- 2) Units of displacements and angles are μm and degree, respectively.
- 3) Error = desired position – measured position + initial position errors.

In order to measure rotational precision, upper platform is moved to $\alpha_p = 97$ deg, $\beta_p = 95.955$ deg, $\gamma_p = 9.20972$ deg rotational position from $\alpha_p = 90$ deg, $\beta_p = 90$ deg, $\gamma_p = 0$ deg when the upper platform is at $x_p = 0$ mm, $y_p = 0$ mm and $z_p = 25$ mm. This motion is repeated 10 times and test results are measured. Test results of rotational motion of hexapod are given in Figure 6.10 and Table 6.4. The table has the desired

position; averages of errors calculated from measured angular positions and maximum deviations from averages.

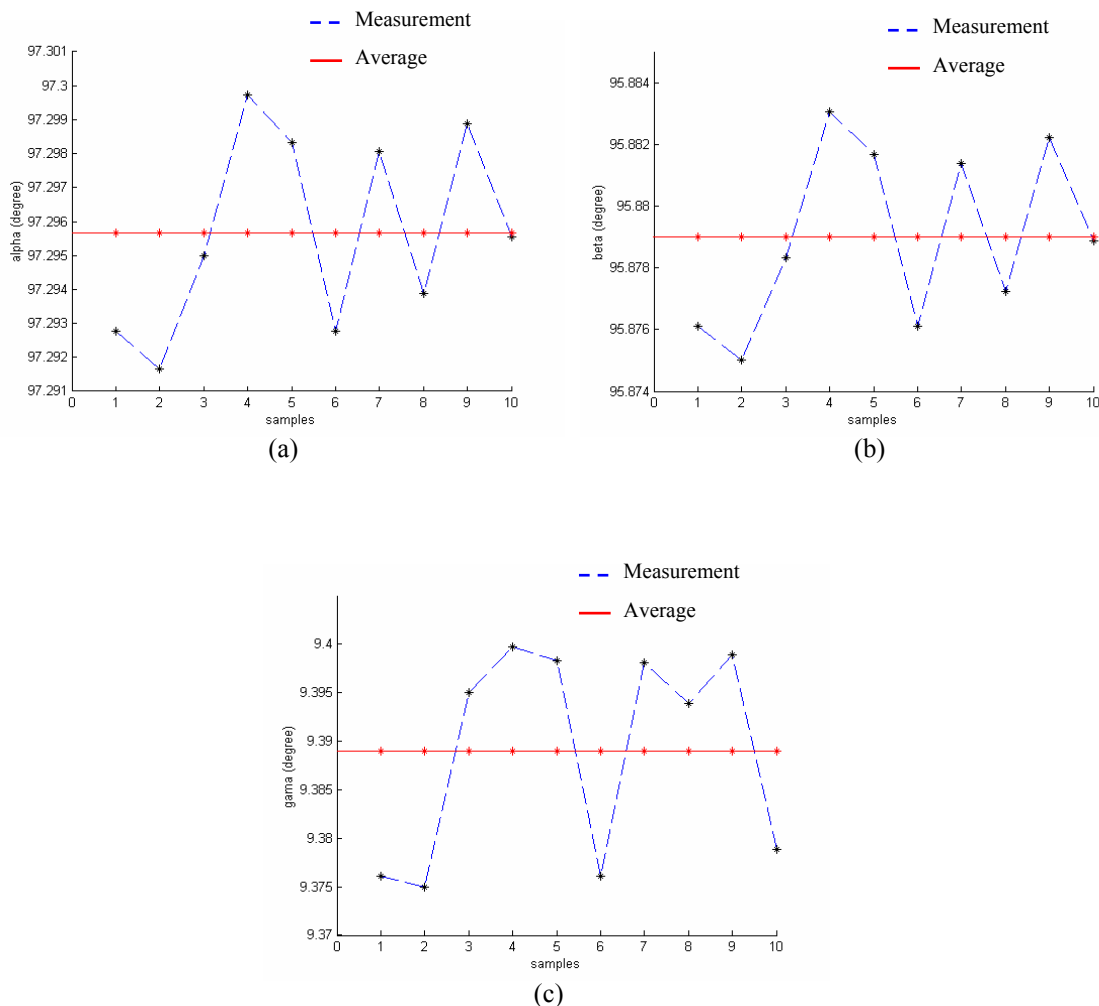


Figure 6.10 (a) α_p , (b) β_p and (c) γ_p measurement results for desired position ($\alpha_p = 97$, $\beta_p = 95.955$ and $\gamma_p = 9.20972$).

Table 6.4 Results of rotational motion.

Desired position ¹ $\alpha_p, \beta_p, \gamma_p$	Averages of errors ^{1,2} $\epsilon_{\alpha}, \epsilon_{\beta}, \epsilon_{\gamma}$	Maximum deviations from averages, respectively ^{1,2}
97 , 95.955 , 9.20972	-0.3239 , 0.1108 , -0.1326	0.004 , 0.004 , 0.014

1) Units of angles are degree.

2) Error = desired position – test result position + initial position errors

As a result, there are initial position errors (Fig. 6.1 and Table 6.1) when all linear motors are fully closed. Fig. 6.2, Fig. 6.3, Fig. 6.4 and Table 6.2 may give an idea about precisions and repeatability, because motions are started from the initial point. However, in those tests linear motors are manually closed by using the program. Therefore, deviations are high in these motions. Limit switches attached to linear motors can be used to fully close the linear motors instead of manual closing. Thus, the upper platform of the hexapod can be set in the same point at the initial position. Therefore, precision and repeatability regarding the motions which are started from the initial position can be improved.

Five different points (Fig. 6.5, Fig. 6.6, Fig. 6.7, Fig 6.8, Fig. 6.9 and Table 6.3) are also tested. Hexapod is moved to every point in a sequence. In addition, rotational motion (Fig. 6.10 and Table 6.4) is tested. Consequence of these tests, repeatability values namely maximum deviations are about desired values. However, measured positions and desired positions are different. This difference can not be considered for motions in a sequence, because errors of previous motion effect actual motion, and errors of actual motion effect following motion. Maximum deviations are important in these motions. It is observed that, the hexapod moves to approximately same positions. This means, hexapod can move right positions after calibrating it.

Making precise holes into which joints are precisely placed, in the joint location points on the platforms can improve initial position errors. Diameters of holes should approximately equal to diameters of bases of joints, in order for precise assembly.

As results of measurements, it is observed that precision of the hexapod is worse than repeatability. This difference can be caused by initial position errors, and initial position errors can be caused by manufacturing errors and assembly errors. Simulations contain no manufacturing and assembly errors. Precision can be increased by simulating the system with initial position errors or making more precise manufacturing and assembly.

CHAPTER SEVEN

CONTROLLING OF AC SERVO MOTOR SYSTEMS

7.1 Introduction

Motors can be controlled by PC-based units or programmable stand-alone units in automation systems. In this chapter, controlling of servo motors is carried out by ADLINK PCI motion control cards. For the experimental study, an experimental rig is constructed and VisualBASIC programs are developed to control simultaneously three OMRON brushless AC servo motors (Omron Corp., 2006) over the cards. In the programs, reference velocity and position curves are taken as inputs, and the parameters of the commands of ADLINK control cards are determined. In this study, servo motors are thought as actuators of a 3-DOF serial manipulator on which the hexapod will be attached.

7.2 Design and Inverse Kinematic Analyses of a 3-DOF Serial Manipulator

Solid models of parts of a 3-DOF manipulator are created parametrically in ABAQUS. Parameters of parts are written in a file whose extension is “py”. Then, by running the script with ABAQUS, solid parts are drawn. The designed manipulator is given in Figure 7.1 (Karagülle et al., 2007).

Solid parts are saved as in the “iges” format. Inverse kinematic analyses are achieved by VisualNASTRAN. “igs” extended solid model files are imported into VisualNASTRAN. Because of API property of the program, assemblies and analyses in it, might be done by VisualBASIC programs and changed rapidly in consequence of parametrical changes (Karagülle et al., 2007).

Parts are constrained in VisualNASTRAN. $p0$ is fixed to ground; revolute motors are described between $p0$ and $p1$, $p1$ and $p2$ and $p2$ and $p3a$. $p3a$ and $p3b$ are rigidly

connected together. Then, orientation and angular velocity meters are assigned to revolute motor constraints to determine angular velocities and angular positions of actuators (Karagülle et al., 2007).

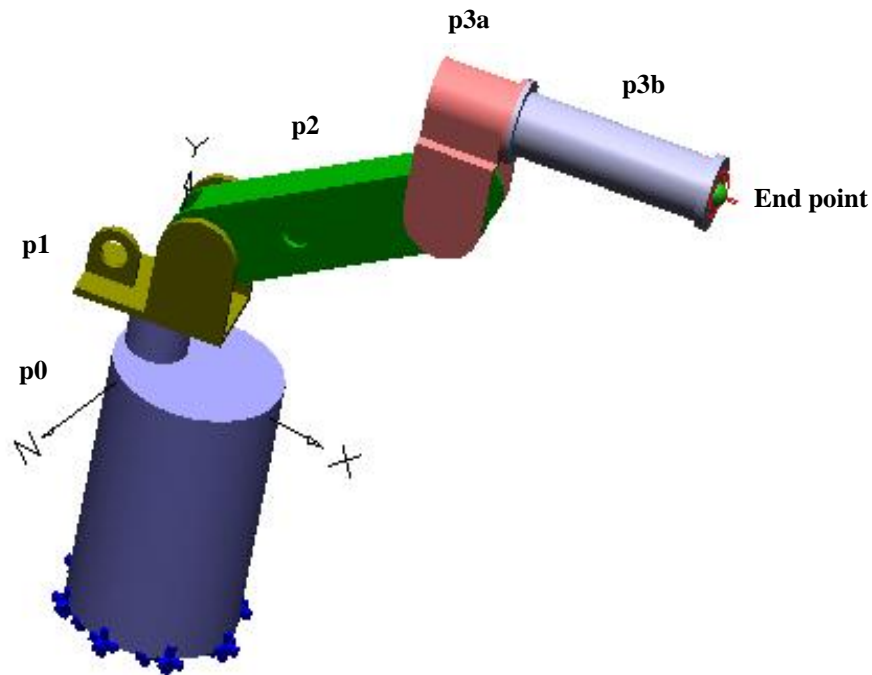


Figure 7.1 The designed 3-DOF serial manipulator.

For inverse kinematic analyses, a weightless part (*pend*) is created at the end of the manipulator. This part is attached rigidly to *p3b* part. Starting and final positions of the end point is determined. Between these points, linear motion is supposed. Velocity inputs of the end point are generated with respect to velocity sinusoid (see Appendix B) (Karagülle et al., 2007).

Generated velocity values are applied to end point as “prescribed motion”. Inverse kinematic analysis is solved by VisualNASTRAN. Results of the solution are angular velocities and angular positions of the actuator. These curves are inputs of control algorithms.

7.3 AC Servo Motor Experimental Rig

7.3.1 The Experimental Rig

AC servo motors are thought as actuators of a 3-DOF serial manipulator. The experimental rig set up with Omron brushless AC servo motors, are shown in Figure 7.2, schematically.

PCI 8132 and PCI 8164 motion control cards of Adlink Company and three items brushless AC servo motors and their servo drivers of Omron Company are used in the experimental rig. The experimental rig which is set up at laboratory is given in Figure 7.3 (a).

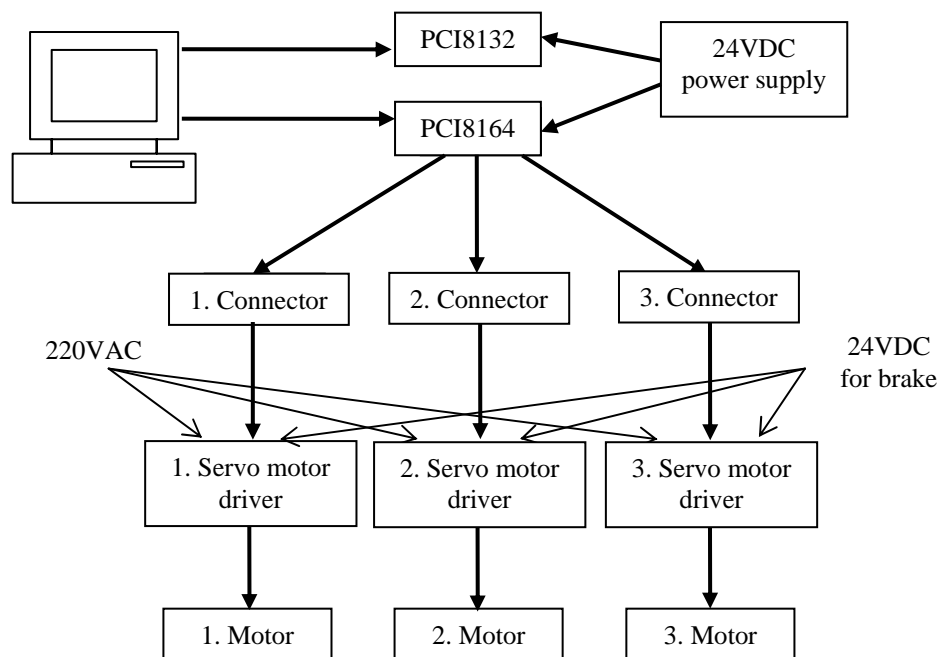


Figure 7.2 Servo motor experimental rig.

Maximum frequency of PCI 8132 is 2.4 million pulses/s and of PCI 8164 is 6.55 million pulses/s. Comprehensive knowledge about the cards are in Appendix A. These cards are placed into a PC. Communication with the cards is achieved over terminal boards. Terminal boards, Figure 7.3 (b), linked to connectors manufactured to connect control cards and servo motor drivers, via bus cables.

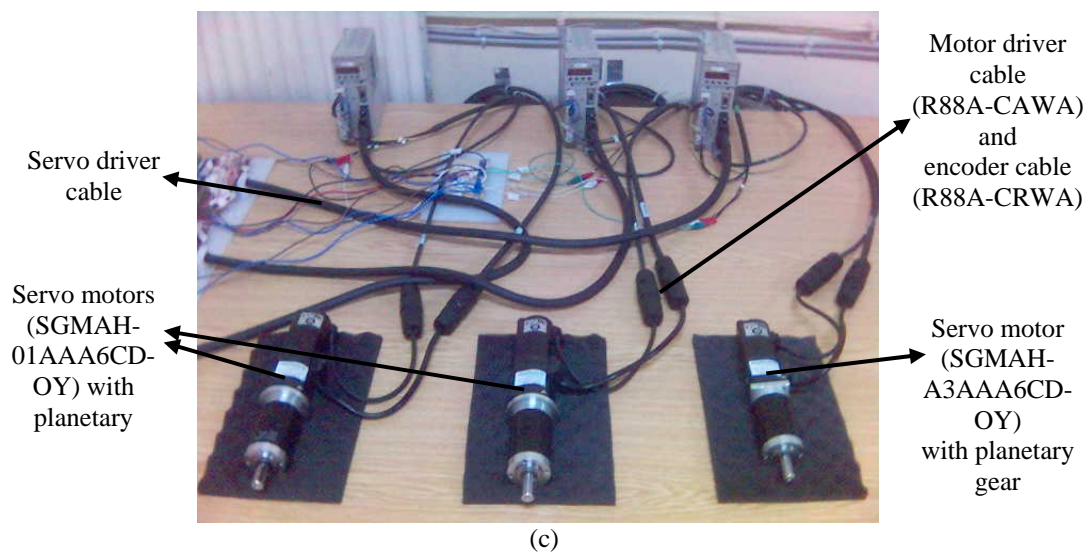
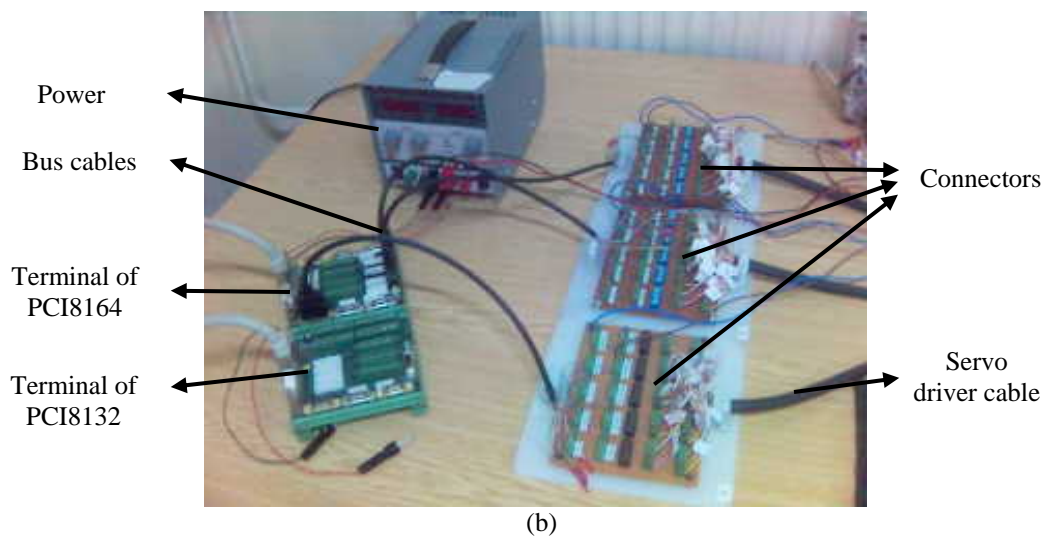
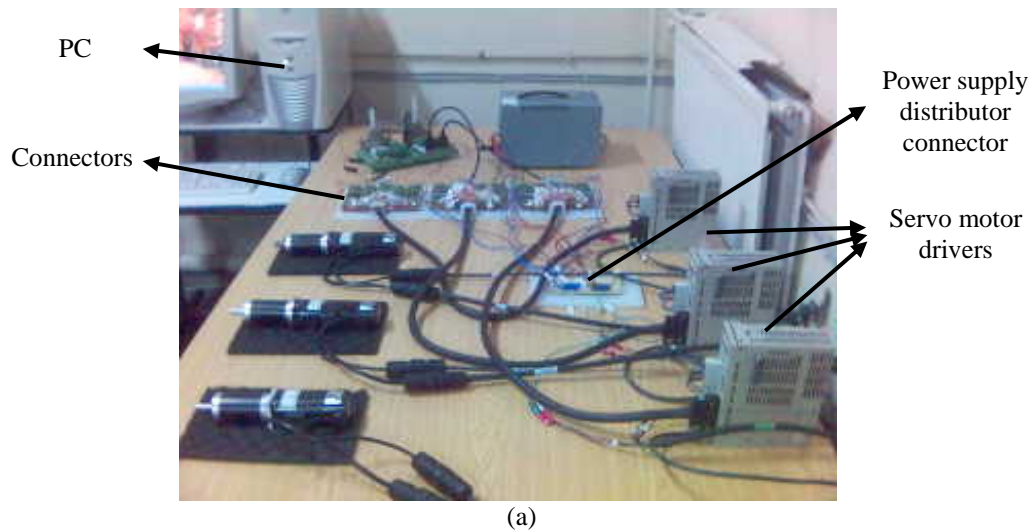


Figure 7.3 (a), (b), (c) The experimental rig.

Omron AC servo motors are shown in Figure 7.3 (c). Technical properties of motors and types of motor drivers are given in Table 7.1. Encoders are serially attached to motors. Servo motor drivers have one motor driving capacity. 203123 type Maxon planetary gears (Maxon Co., 2006) whose reduction ratio is 74 are assembled to servo motors. Flanges and pinion gears are designed and manufactured in order to assemble motors and gears.

Table 7.1 Properties of servo motors.

Motor Number	Motor Type	Voltage (VAC)	Current (A)	Power (W)	Torque (Nm)	Revolution (rev/min)	Encoder type	Encoder resolution (2048 x 4 p/rev)	Servo motor driver type
1	SGMAH – 01AAA6CD – OY	220	0.91	100	0.318	3000	Incremental	13 bite	SGDH – 01AE–OY
2	SGMAH – 01AAA6CD – OY	220	0.91	100	0.318	3000	Incremental	13 bite	SGDH – 01AE–OY
3	SGMAH – A3AAA6CD – OY	220	0.44	30	0.095	3000	Incremental	13 bite	SGDH – A3AE–OY

7.3.2 Connections of Servo Motors and Drivers

Connections between AC servo motors, AC servo motor drivers and a PC are realized as Figure 7.4. R88A – CAWA type cables make connections between drivers and motors. Encoder feedback signals are sent to drivers via R88A – CRWA type cables.

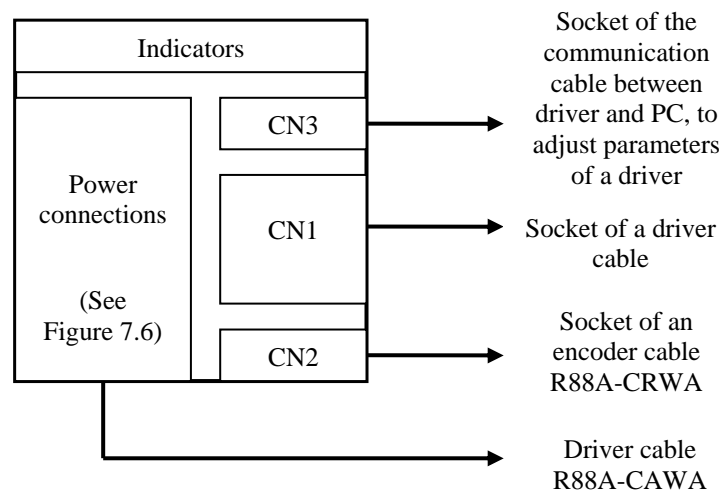


Figure 7.4 Front panel view of a motor driver.

Motion control cards are powered with 24VDC via their *CNI* socket. To connect servo motor drivers to motion control cards, a connector is designed. Bus cables are used between motion control cards and the manufactured connector. Pin assignment of a bus cable is shown in Figure 7.5 and data of these pins are given in Table 7.2.

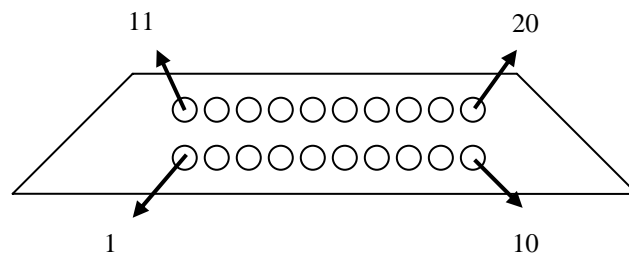


Figure 7.5 Pin assignments of bus cables.

Output signals to drive motors are *OUT +*, *OUT -*, *DIR +* and *DIR -*.in Table 7.2. These signals are motor velocity (frequency) adjustment signals (*OUT +*, *OUT -*) and motor direction adjustment signals (*DIR +*) and (*DIR -*), respectively. Input signals to take feedback from encoder are *EA +*, *EA -*, *EB +*, *EB -*, *EZ +*, *EZ -*. These signals are output signals of A phase (*EA +*) and (*EA -*), B phase (*EB +*) and (*EB -*) and Z reference position signals (*EZ +*) and (*EZ -*) of encoders, respectively.

Table 7.2 Values of bus cable.

Bus cable		CNA 1 - 4	Bus cable		CNA 1 - 4
Pin numbers	Colour	Signal	Pin numbers	Colour	Signal
1	Brown	IGND	11	Brown (-)	Empty
2	Red	DIR +	12	Red (-)	DIR -
3	Pink	OUT +	13	Pink (-)	OUT -
4	Yellow	Empty	14	Yellow (-)	Empty
5	Green	EZ +	15	Green (-)	EZ -
6	Blue	EA +	16	Blue (-)	EA -
7	Light blue	EB +	17	Light blue (-)	EB -
8	Purple	ERC	18	Purple (-)	INP
9	Gray	+24V (Out)	19	Gray (-)	RDY
10	White	IGND	20	White (-)	IGND

Bus cables come from *CNA* output sockets of the PCI 8164 motion control card are connected to servo motor driver cables come from *CNI* output socket of drivers, via manufactured connectors. These connections are given in Table 7.3.

OUT +, *OUT -*, *DIR +*, *DIR -* pins of bus cables (Table 7.3) are wired on the connector to *+ CW*, *- CW*, *+ CCW*, *- CCW* pins of driver cables, respectively. In the same way, *EA +*, *EA -*, *EB +*, *EB -*, *EZ +*, *EZ -* pins are wired to *+ A*, *- A*, *+ B*, *- B*, *+ Z*, *- Z* pins, respectively. Besides, to change mode of the drivers “sleep” to “run”, 24 VDC and ground of the power supply are connected to *+24VIN* and *RUN* pins, respectively.

Table 7.3 Cables connected on the connector.

Driver cable			Bus cable		
Cable numbers	Colour	Signal	Cable numbers	Colour	Signal
7	Gray / red (-)	+CW	3	Pink	Out +
8	Gray / black (-)	-CW	13	Pink (-)	Out -
11	Yellow / red (-)	+CCW	2	Red	Dir +
12	Yellow / black (-)	-CCW	12	Red (-)	Dir -
19	Gray / red (--)	+Z	5	Green	EZ +
20	Gray / black (--)	-Z	15	Green (-)	EZ -
33	Orange / red (---)	+A	6	Blue	EA +
34	Orange / black (---)	-A	16	Blue (-)	EA -
35	Gray / black (---)	-B	17	Light blue (-)	EB -
36	Gray / red (---)	+B	7	Light blue	EB +
40	Pink / black (---)	RUN	Power supply	Ground	
47	Gray / red (----)	+24VIN		+24VDC	

Motor drivers are prepared by wiring power cables and driver cables (R88A – CAWA) on the front panels as shown in Figure 7.6. It is important to correctly connect *U*, *V*, *W* phases and grounds of power cables to the front panels and 24 VDC to *B* pins of the motor cables. 24VDC releases mechanical brakes of the motors.

Motor encoder cables (R88A – CRWA) are connected to CN2 sockets (Fig. 7.4) on the front panel of drivers. Therefore, taking feedback signals is possible. Motor encoder cables and motor driver cables are attached to their sockets on the motors.

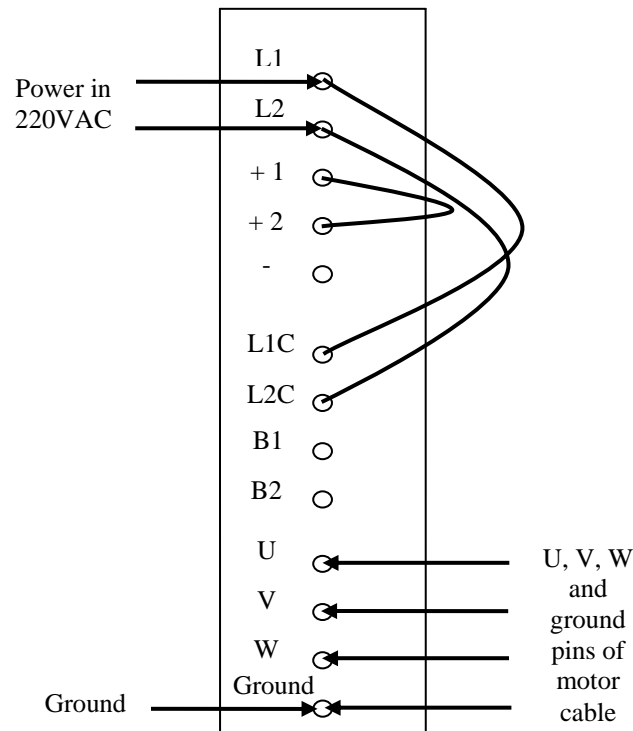


Figure 7.6 Power connections of a driver.

7.3.3 Adjusting Servo Motor Parameters

The CN3 socket (Fig. 7.4) of a driver is lined to a COM port of a PC via a RS communication cable. Necessary parameters are adjusted and loaded with WMON WinE Version 2.0 program of the drivers. Parameters are separately loaded to drivers. Program controls only one servo motor.

Drivers have to be opened and motor brakes have to be released before running the program and adjusting parameters. Firstly, rigidity of the motor has to be adjusted. Rigidity has to be suitable for working environment. If the rigidity is adjusted higher than necessity, motor shaft makes vibrations. Besides, motor current and voltage are adjusted by auto tuning over the program. Then parameters are

adjusted with respect to Table 7.4. Parameters which are different from factory settings are saved by selecting “save adjustments”.

Table 7.4 Adjusted parameters of servo motor drivers.

		SGDH-01AE-OY type servo motor driver	SGDH-A3AE- OYtype servo motor driver
Parameters	Explanations	Value	Value
Pn000	Direction and control type selection	0010H	0010H
Pn001	Alarm types	1002H	1000H
Pn002	Velocity and torque control and feedback selection	0000H	0000H
Pn003	Analogue control	0002H	0002H
Pn100	Speed loop responsiveness	30	20
Pn101	Speed loop integral time constant	3000	4500
Pn102	Position loop responsiveness	30	20
Pn103	Inertia ratio	1180	962
Pn104	Speed loop responsiveness 2	40	80
Pn105	Speed loop integral time constant 2	2000	2000
Pn106	Position loop responsiveness	40	40
Pn107	Bias rotational speed	1	1
Pn10B	Speed control settings	0004H	0004H
Pn10F	P control switching	10	10
Pn110	Online auto-tuning setting	3110H	3110H
Pn200	Position control setting	0010H	0010H
Pn201	Encoder divider rate	16384	16384
Pn202	Electronic gear ratio (numerator)	4	4
Pn203	Electronic gear ratio (denominator)	1	1
Pn207	Position control setting	0000H	0000H
Pn50A	Input signal selection 1	8100H	8100H
Pn50B	Input signal selection 2	6548H	6548H

These parameters can be saved in an “usr” extended file. If there is no parameter file available, parameters have to be adjusted according to Table 7.4. Furthermore, these parameters can be adjusted by using the front panels of the drivers.

Precision of the motor encoders is 2048. So, 2048 pulses have to be sent over the PCI control cards to rotate motor shafts for one revolution; however, this value can be changed by adjusting Pn202 and Pn203 parameters (Table 7.4). For example, if Pn202 / Pn203 is 8 (originally 4), then the requested pulses are $2048 \times 2 = 4096$.

7.4 Controlling of AC Servo Motors

It is possible to program ADLINK control cards with VisualBASIC. PCI 8132 and PCI 8164 components are added into the project and onto the form of the project. PCI8132.bas and PCI8164.bas modules, which come with the installation CD of the motion control cards, are also added to project. These modules contain commands which are not recognized by VisualBASIC (see Chapter 5.2.1).

A VisualBASIC based test program is developed to control one servo motor uniquely via PCI 8132 card. This program is shown in Figure 7.7. In this program, there are all motion types that PCI 8132 control card has. When selecting a motion profile on the program, required parameter input boxes on the form are colorized with different colours. Thus, parameter input boxes irrelevant to the selected motion profile are locked. Different motion types are tested via PCI 8132 card with respect to open loop control.

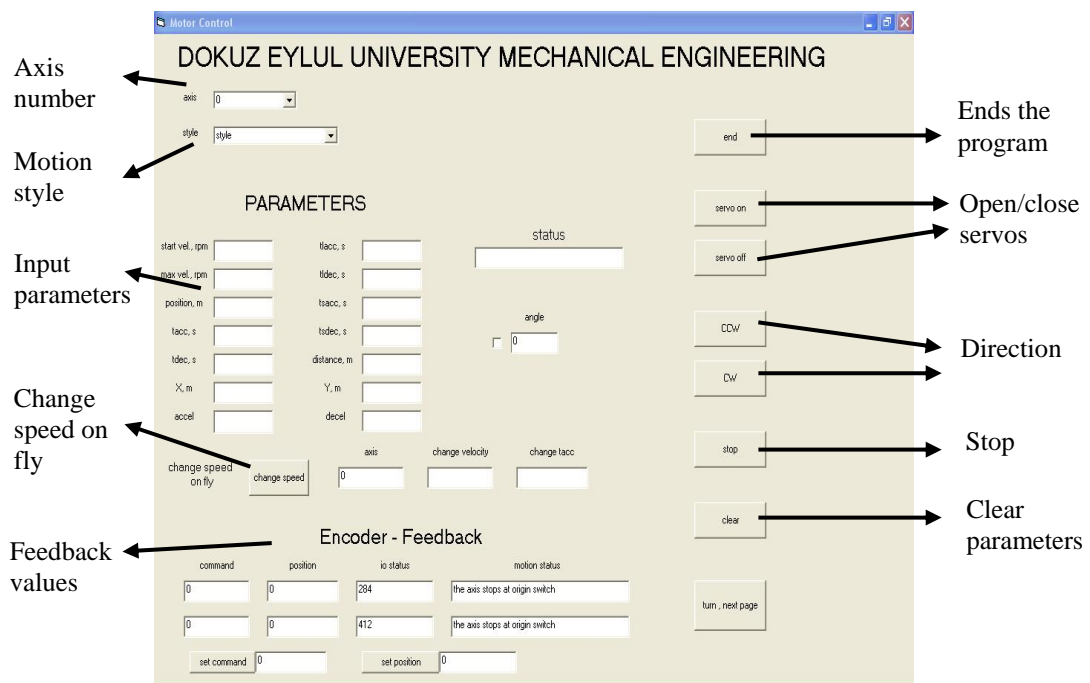


Figure 7.7 The test program developed to control one servo motor with open loop control.

Position input must be pulse numbers and velocity input must be pulse numbers per a second; however, for comprehensibility, position is taken in meters (or degrees) and velocity is taken in revolution per minute. Position and velocity values of motors are 2048 pulse/rev and 2048 pulse/s, respectively. These inputs are calculated as following:

$$p = \frac{2048}{360} n_{gear}.pd \quad (7.1)$$

$$p = \frac{2048}{2\pi.r_0} n_{gear}.pm \quad (7.2)$$

$$v = \frac{2048}{60} n_{gear}.vd \quad (7.3)$$

Where, p is position value whose unit is pulse, pd is angular position of output shaft of the gear attached to the motor (degree), pm (meter) is linear movement of output shaft of the gear, v is velocity value (pulse/s), vd is angular velocity of output shaft of the gear (rev/min), n_{gear} is reduction ratio, r_0 (m) is radius of shaft or pulley etc. attached to the shaft of gear.

Closed loop control of two servo motors is tested by a different developed VisualBASIC program. This program is given in Figure 7.8. Reference curves which are wanted servo motors to follow simultaneously are sent to motors over the program. Equations (7.1), (7.2) and (7.3) are also used in this program. T (trapezoidal) motion is used as motion profile. T motion profiles are sent to motors one after another with respect to a time interval Δt . Error signals are generated by taking feedback. New positions are calculated according to closed loop control. Motors follow the curves with acceptable errors and at the right time. Command of T motion profile is:

$$B_8164.StartTAMove(axis, pos, svel, mvel, tacc, tdec) \quad (7.4)$$

Parameters of this command are: *axis* is the working axis number, *pos* is position value (pulse), *svel* (pulse/s) is starting velocity, *mvel* (pulse/s) is maximum velocity, *tacc* and *tdec* are acceleration and deceleration times in terms of second. In this program total motion time and time interval are taken as 20 s and $dt = 0.05025$ s, respectively. Formulation of reference curves are given as following equation:

$$\theta_i = 360 \sin\left(\frac{\pi}{20} it\right), i = 1, 2 \quad (7.5)$$

Where, θ_i and t are in units of degree and second, respectively. Equation (7.1) is used to change degree to pulse numbers.

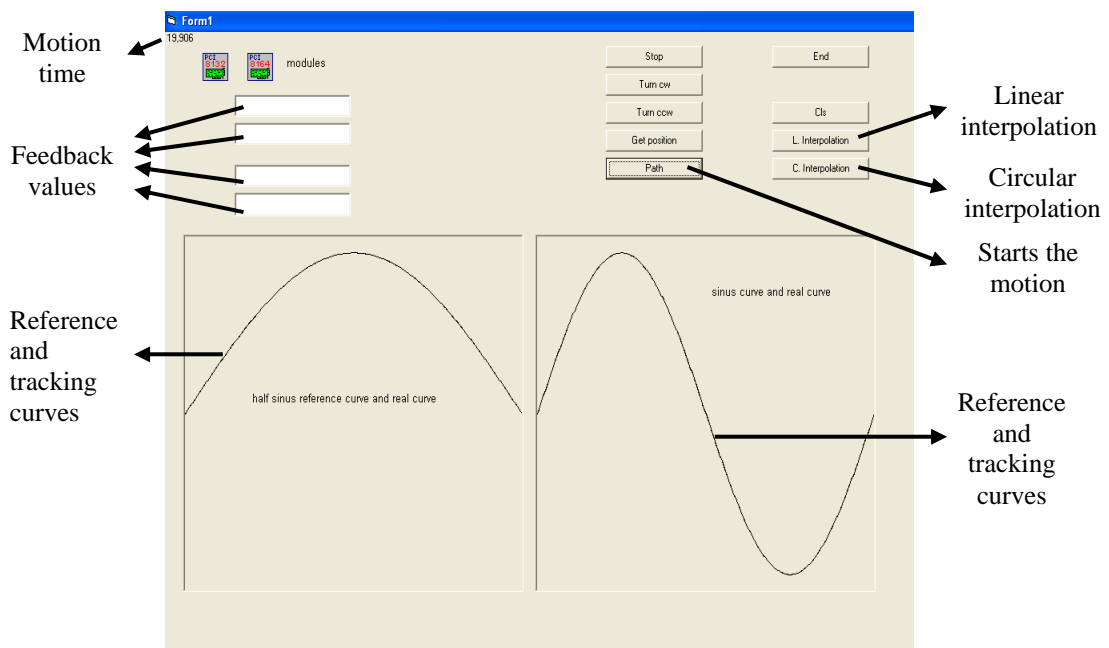


Figure 7.8 Closed control program of two servo motors.

Tracking curves successfully follow the reference curves by using this program (Fig.7.8), however, motor shafts vibrate because of using full T motion profiles in a sequence. Motors accelerate and decelerate in all motion steps of the sequence.

Another VisualBASIC program is developed to control simultaneously three AC servo motors for open loop and closed loop control principles. This program is shown in Figure 7.9. The entire codes of the developed VisualBASIC program are in

Appendix E. In this program different control methods and algorithms are carried out. The end point of the manipulator is moved linearly from (0.51171, 0.3705, 0) point to (0, 0.5, 0.3) point at simulation which is in Section 7.2. Results whose inputs are angular position – time curves which are generated from inverse kinematic analyses are presented. Total motion time is 5 s, and number of samples is $ns = 41$.

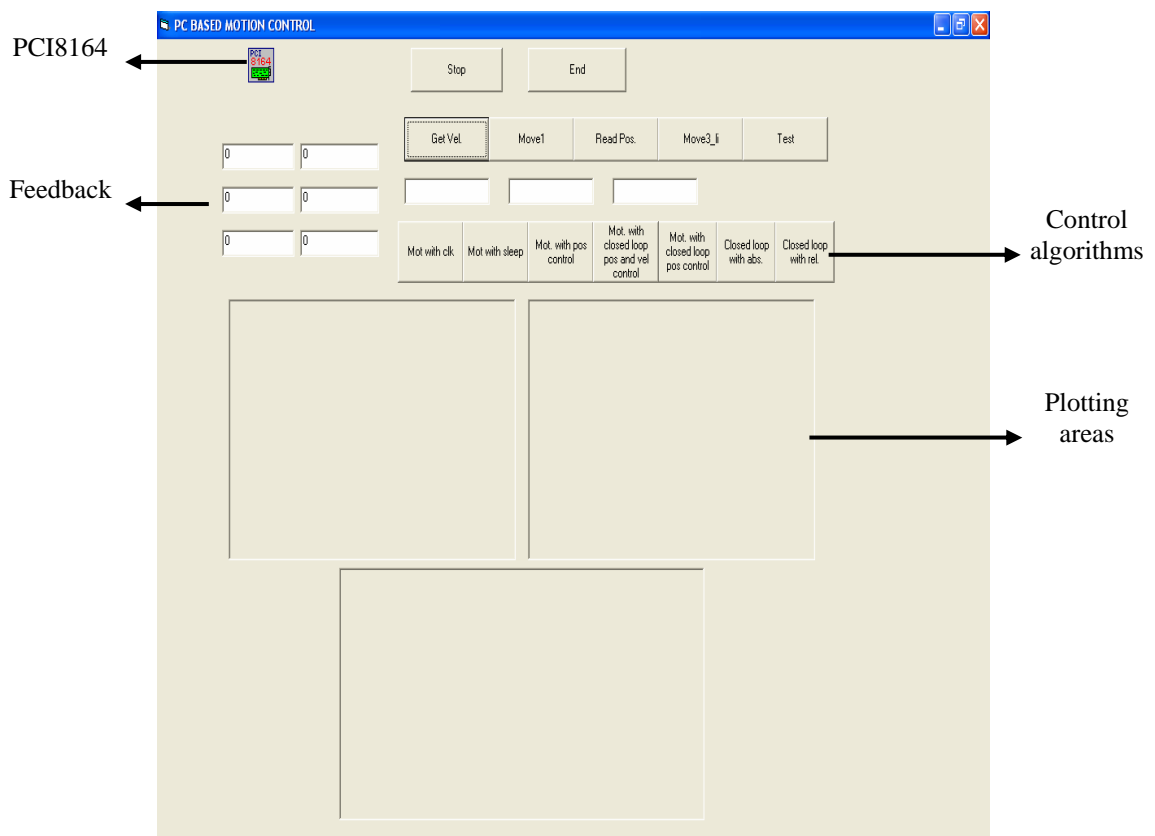
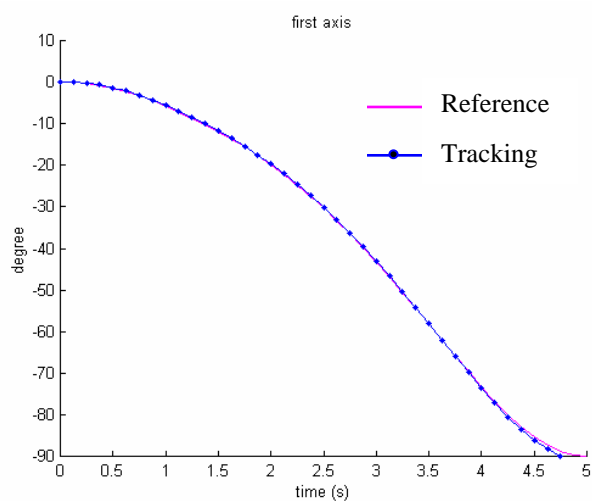


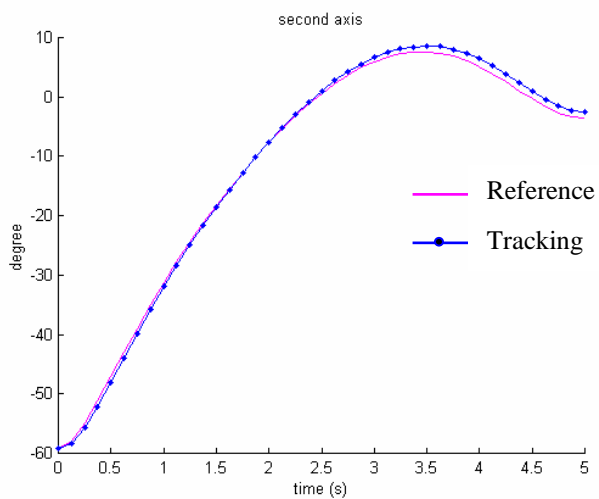
Figure 7.9 Control program of three servo motors with different algorithms.

Fundamental commands for open loop control whose input is velocity (first and second algorithms) are given in Table 7.5. In these algorithms, motors change their starting velocities to second step velocities in the time interval dt with first command. Then with a loop, maximum velocities are changed in time interval dt by means of second command. Time constraint is achieved by internal counter of the card in first algorithm and “sleep” command in second algorithm. Results are exhibited in Figure 7.10 and Figure 7.11.

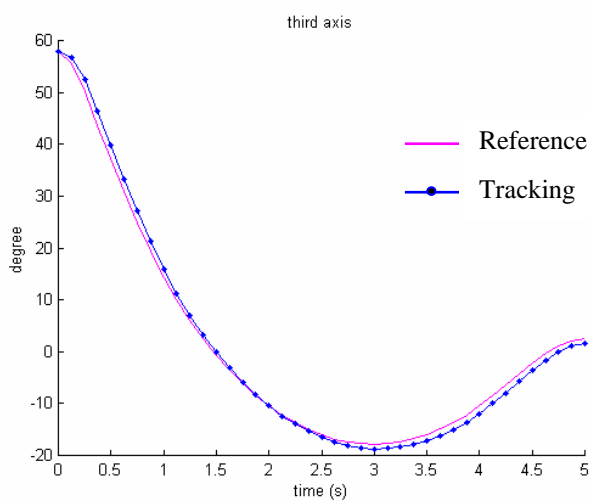
Table 7.5 Control algorithms.

Control method	Input	Algorithm number	Fundamental commands	Property
Open loop	Velocity	1	$B_8164.TVMove(axis, svel, mvel, dt)$ $B_8164.VChange(axis, mvel, dt)$	Time constraint is achieved by internal counter. Acceleration time is equal to the sampling time.
		2		Time constraint is achieved by "sleep" command. Acceleration time is equal to the sampling time.
	Position	3	$B_8164.StartTAMove(axis, pos, svel, mvel, tacc, tdec)$	
Closed loop	Position	4	$kperr(k) = kp*(ang(k+1) - pos(k-1))$ $kverr(k) = npuls(k+1)$ $B_8164.StartTAMove (axis, kperr(k), npuls(k), kverr(k), dt, 0)$	Acceleration time is equal to the sampling time.
	Position and velocity	5	$kperr(k) = kp*(ang(k+1) - pos(k-1))$ $kverr(k) = kv*(npuls(k+1) - v(k-1))$ $B_8164.StartTAMove (axis, kperr(k), npuls(k), kverr(k), dt, 0)$	
	Position	6	$kperr(k) = k*(ang(k) - pos(k - 1))$ $kang(k) = ang(k + 1) + kperr(k)$ $B_8164.StartTAMove (axis, kang(k), npuls(k), npuls(k+1), dt, 0)$	
		7	$kperr(k) = k*(ang(k) - pos(k - 1))$ $kang(k) = ang(k + 1) - ang(k) + kperr(k)$ $B_8164.StartTRMove (axis, kang(k), npuls(k), npuls(k+1), dt, 0)$	



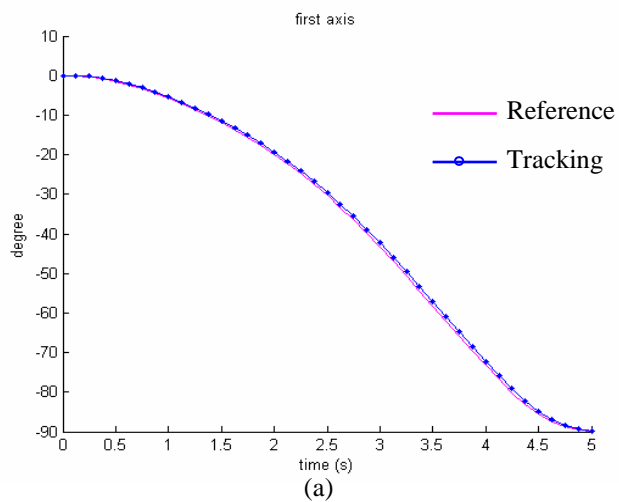


(b)



(c)

Figure 7.10 (a) the first axis, (b) the second axis, (c) the third axis results for the first algorithm.



(a)

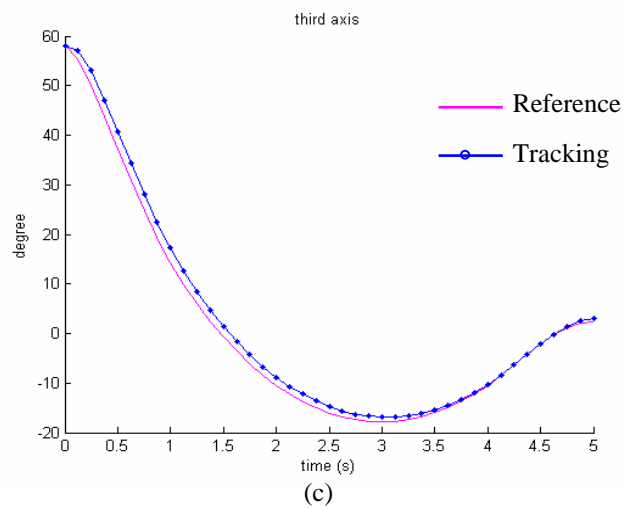
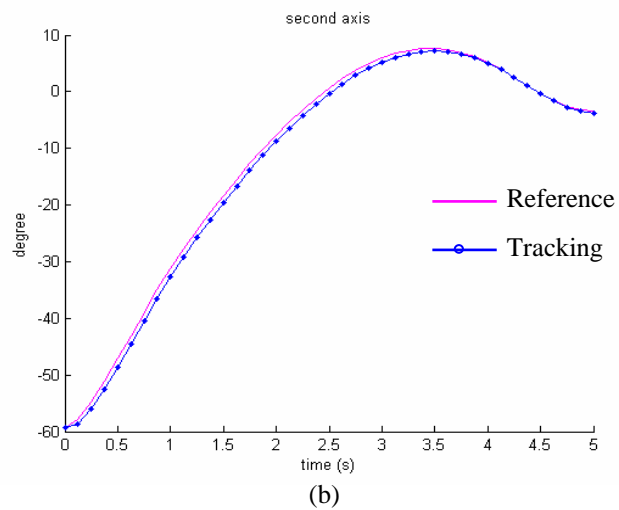
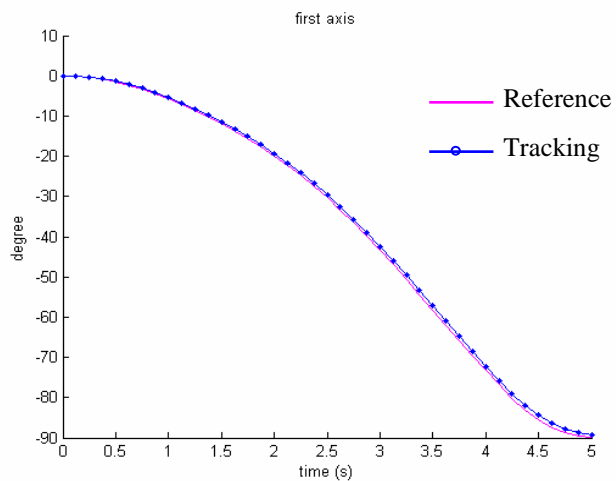
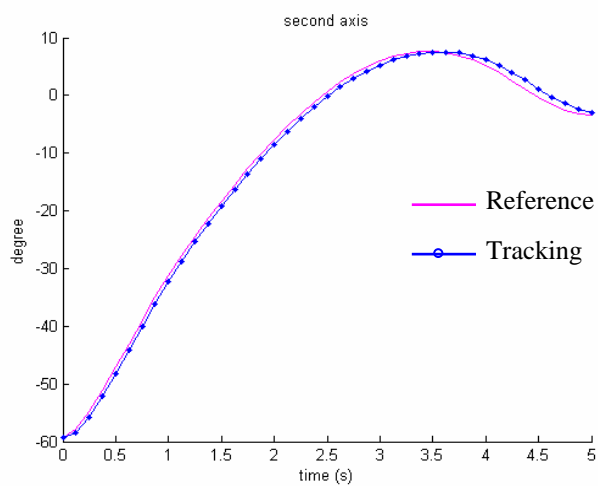


Figure 7.11 (a) the first axis, (b) the second axis, (c) the third axis results for the second algorithm.

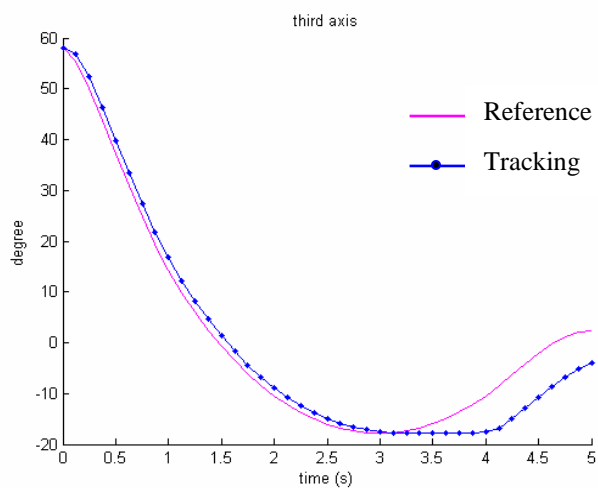
Fundamental command for open loop control whose input is position (third algorithm) is given in Table 7.5. In this algorithm, sampling time dt is equal to acceleration time and the command is sequentially sent to motors with a time interval dt . Time constraint is achieved by internal counter of the card. Results are presented in Figure 7.12.



(a)



(b)



(c)

Figure 7.12 (a) the first axis, (b) the second axis, (c) the third axis results for the third algorithm.

A block diagram of the closed loop control which is used in the fourth and the fifth algorithms is given in Figure 7.13. In the block diagram, inputs are position in terms of pulses and velocity in terms of pulse/s; outputs are angular position and angular velocity. Where, kp and kv are position gain coefficient and velocity gain coefficient, respectively; ang is position input comes from inverse kinematic analysis, $npuls$ is velocity input, pos is position feedback, v is velocity feedback, $axis$ is axis number, $kperr$ and $kverr$ are respectively position and velocity inputs. $kv = 1$ and $v = 0$ are taken for only position inputted control. Fundamental commands of these control algorithms are given in Table 7.5. Results are presented in Figure 7.14 and Figure 7.15 for the fourth and the fifth algorithms respectively.

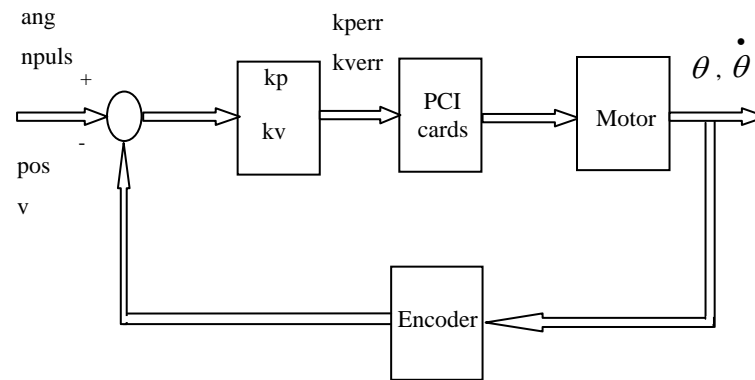
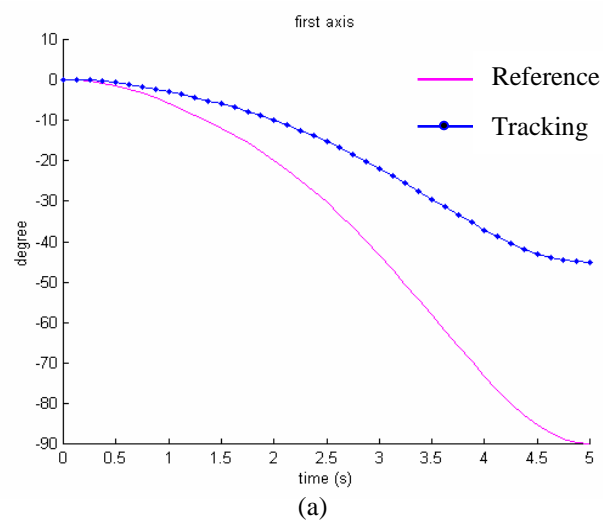


Figure 7.13 Closed loop block diagram for the fourth and the fifth algorithms.



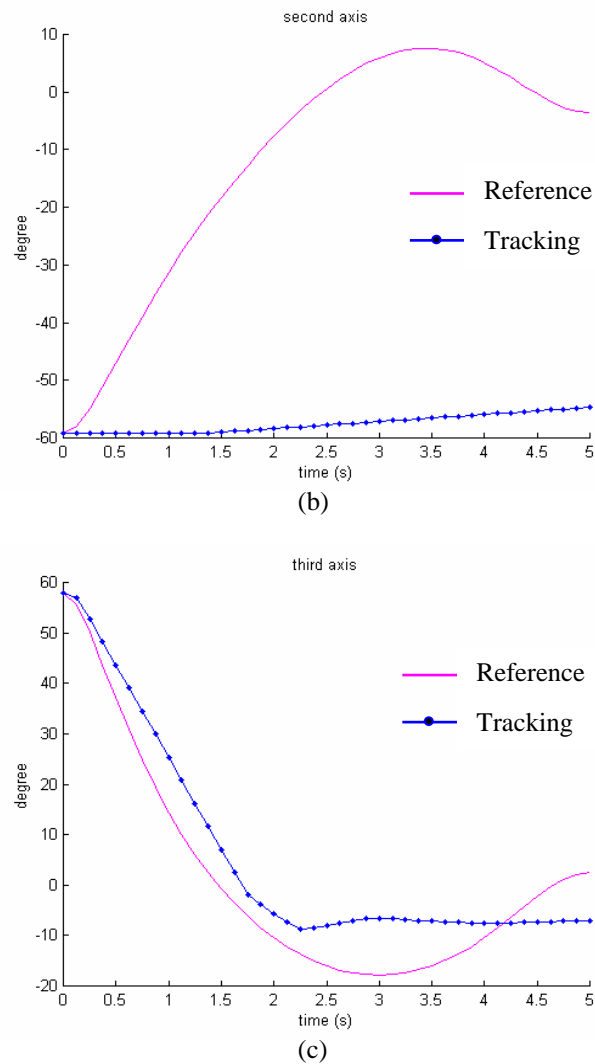


Figure 7.14 (a) the first axis, (b) the second axis, (c) the third axis results, for $k_p = 0.7$ and $k_v = 1$, for the fourth algorithm.

The sixth and the seventh algorithms are offered because of instability of the system occurred by the fourth and the fifth algorithms. Block diagram of the closed loop control which is used in the sixth and the seventh algorithms is given in Figure 7.15. Fundamental commands of these control algorithms are given in Table 7.5. Absolute motion (*TAMove*) is used in the sixth algorithm and relative motion (*TRMove*) is used in the seventh algorithm. Reference point is taken a constant point for absolute motion and finishing point of previous motion for relative motion. k is gain coefficient and $kang$ is position input. It is necessary to take $ka = 1$ for *TRMove*

and $ka = 0$ for *TAMove*. Results of these algorithms are in Figure 7.16 for $k = 0.5$, in Figure 7.17 for $k = 0.7$ and in Figure 7.18 for $k = 0.9$.

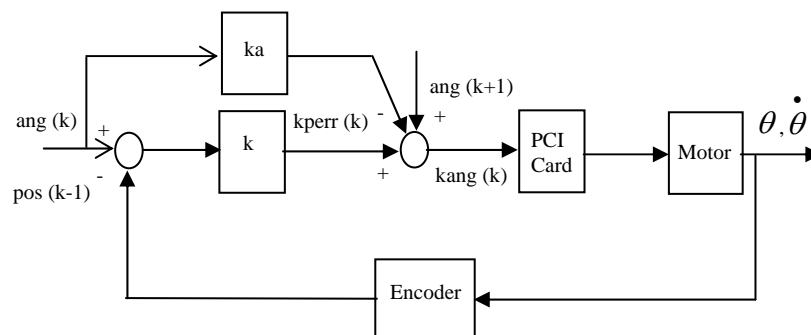
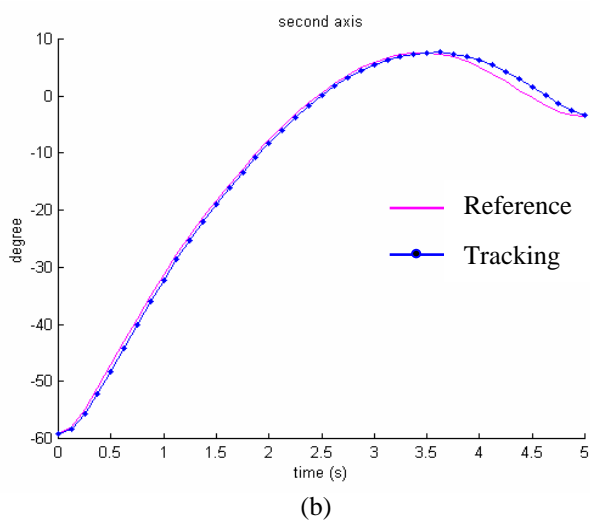
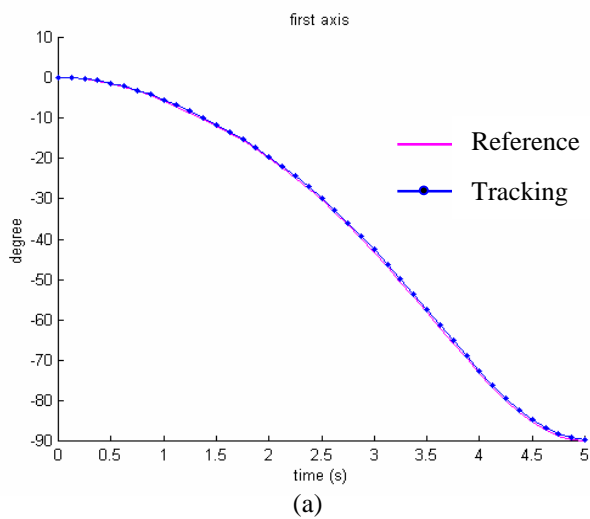


Figure 7.15 Closed loop control block diagram for the sixth and the seventh algorithms



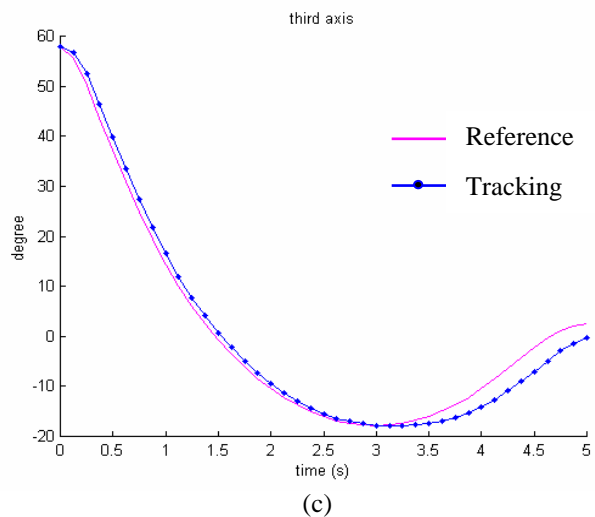
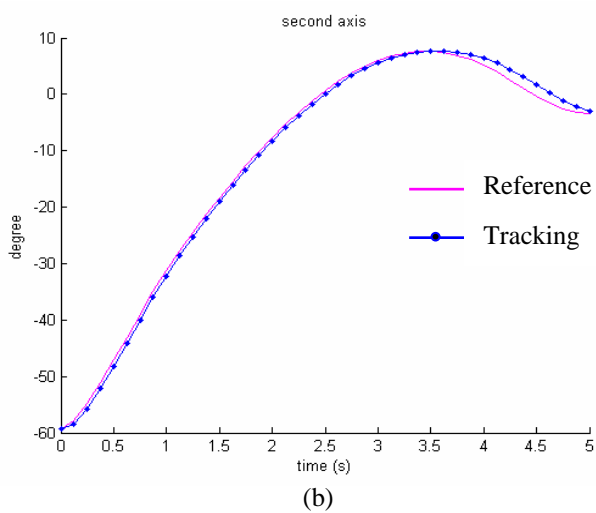
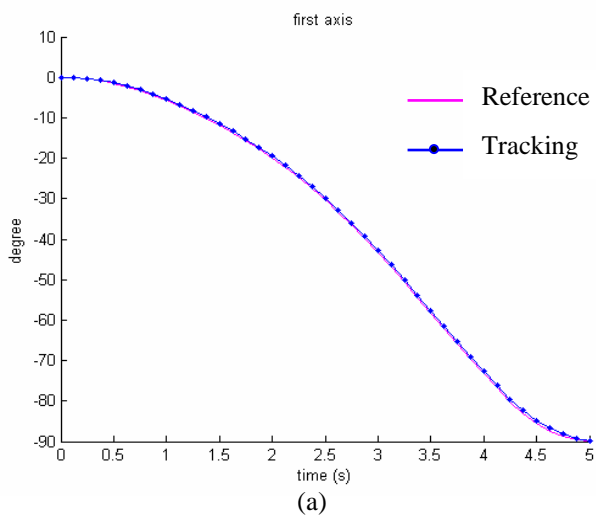


Figure 7.16 (a) the first axis, (b) the second axis, (c) the third axis results for $k = 0.5$ for the sixth and the seventh algorithms.



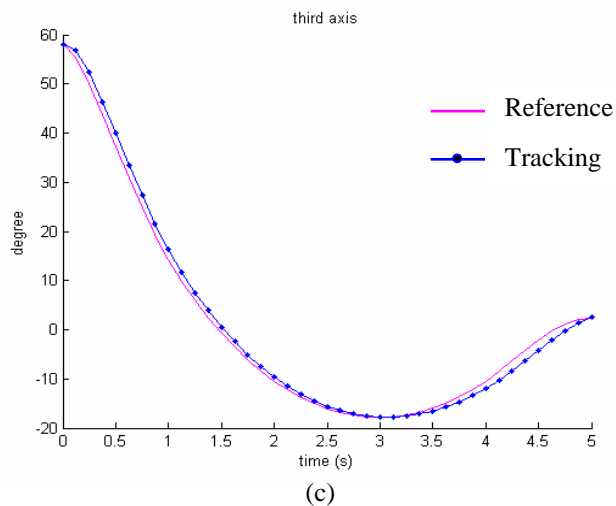
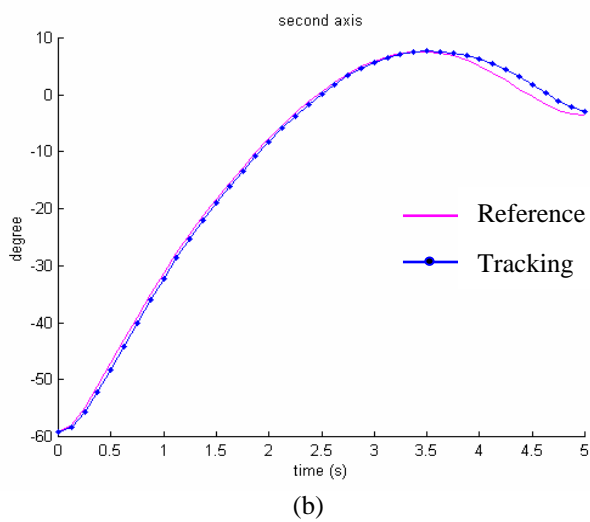
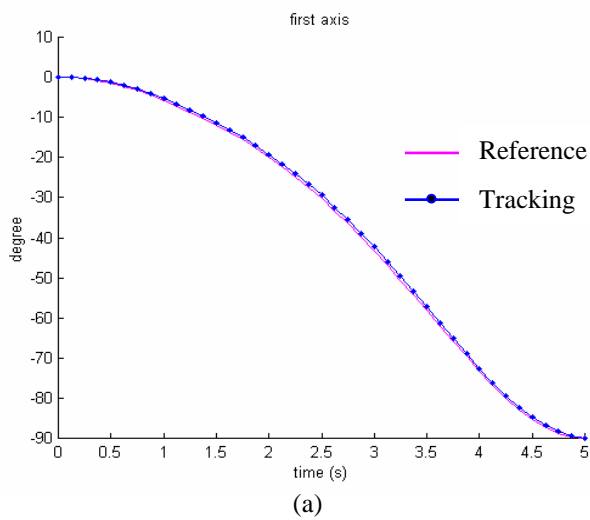


Figure 7.17 (a) the first axis, (b) the second axis, (c) the third axis results for $k = 0.7$ for the sixth and the seventh algorithms.



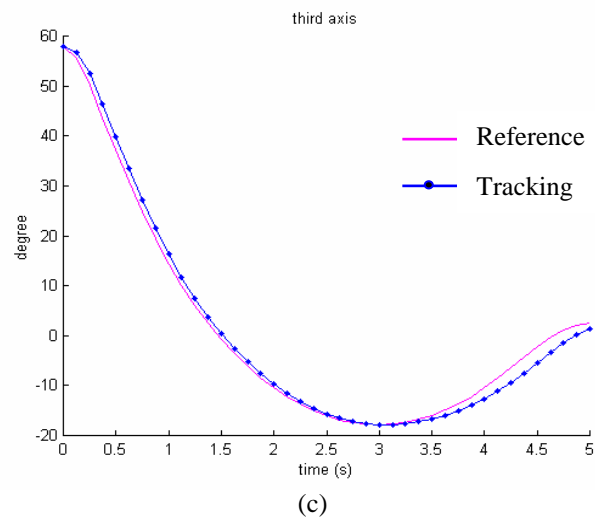


Figure 7.18 (a) the first axis, (b) the second axis, (c) the third axis results for $k = 0.9$ for the sixth and the seventh algorithms.

7.5 Results of the Controlling Servo Motor Systems

In this chapter, an experimental rig is set and inverse kinematic analyses are done. PC-based motion control of motors is realized according to the outputs of the analyses.

For the motor control system, it is observed that open loop control responses very well and the tracking curves resulted from closed loop control algorithms follow the reference curves with acceptable errors. Thus, errors are minimized.

For closed loop control, it is observed that the sixth and the seventh algorithms (see Table 7.5) whose block diagram is shown in Figure 7.15 give more accurate results (Fig.7.16, Fig. 7.17 and Fig. 7.18) than the fourth and the fifth algorithms (see Table 7.5), whose block diagram is given in Figure 7.13. The sixth and the seventh algorithms prevent instabilities that come into existence compared to the fourth and the fifth algorithms (Fig. 7.14). It is studied that system gives good responses in 0.5 – 0.9 interval of the gain coefficient (k) and the best responses when the gain (k) is 0.7 (Fig.7.16, Fig.7.17 and Fig.7.18).

In position inputted algorithms shown in Table 7.5, dt time interval is equated to acceleration time for T motion profile and next motion is sent after dt , instead of using full T motion profile in a sequence (Fig. 7.8). Thus, vibrations on the shaft of the motors are reduced and smoother motions are obtained.

It is observed that using the internal counters of the cards as timer for time constraint makes better responses than using “sleep” command in algorithms. In the end of motions, motors are successfully followed the reference curves.

Tracking curves are generated by taking feedback values from the encoders attached to the rears of servo motors. Closed loop control tracking curves can be improved by taking feedback values from external encoders attached to the shafts of gears instead of using the encoders of servo motors.

For all algorithms, it is accomplished that motions finish in the time constraint which is given.

CHAPTER EIGHT

CONCLUSIONS

A six degree of freedom parallel robot manipulator called hexapod is discussed in order for PC-based control of hexapod. A design is made as a result of inverse kinematic analyses. Standard parts are determined and supplied. Other parts are manufactured after creating 2D and 3D drawings. A control panel is created and connections are made.

Analyses are done with VisualNASTRAN 4D. VisualNASTRAN is controlled by developed VisualBASIC programs. Created solid parts are imported into VisualNASTRAN. Inverse kinematic analyses whose inputs are positions and orientations of the movable upper platform are solved. Lengths of linear motors are found. Point – to – point open loop control is applied to the hexapod by using lengths of linear motors. ADLINK PCI motion control cards are used to drive linear motors.

Obtained results of hexapod according to specific motions and initial position errors are measured by the CMM. As results of measurements, it is observed that precision of the hexapod is worse than repeatability as seen in Chapter 6. This difference can be caused by initial position errors, and initial position errors can be caused by manufacturing errors and assembly errors. Simulations contain no manufacturing and assembly errors. Precision can be increased by simulating the system with initial position errors or making more precise manufacturing and assembly.

Repeatability values obtained from motions which start from initial position are worse than repeatability values obtained from motions which start from any positions in the workspace. Because, the initial position is set by closing linear motors manually. Limit switches attached to linear motors can be used to fully close the linear motors instead of manual closing. Thus, the upper platform of the hexapod can be set in the same point at the initial position; and precision and repeatability regarding the motions which are started from the initial position can be improved.

Making precise holes into which joints are precisely placed in the joint location points on the platforms can decrease initial position errors. Diameters of holes should approximately equal to diameters of bases of joints, in order for precise assembly.

Besides, hexapods are very expensive robots in the market. It is achieved that the hexapod is created much cheaper than the commercial hexapods which are sold in the market.

In addition, brushless AC servo motor systems on which the hexapod will be attached are also examined. In order to control motors, a three degrees of freedom serial manipulator is designed. Solid models of the manipulator are created. Solid models are used in VisualNASTRAN for analyses. Inverse kinematic analyses are done with respect to specific motions. An experimental rig is developed to test the system. VisualBASIC programs are developed to control brushless AC servo motors. Different algorithms are tested. Outputs of analyses are used as inputs of control algorithms. As a result of testing, appropriate control methods and algorithms are determined (Chapter 7).

For the motor control system, it is observed that open loop control responses very well and the tracking curves resulted from closed loop control algorithms follow the reference curves with acceptable errors. Thus, errors are minimized. For closed loop control algorithms, the sixth and the seventh algorithms prevent instabilities that come into existence compared to the fourth and the fifth algorithms (Fig. 7.14). It is studied that system gives good responses in 0.5 – 0.9 interval of the gain coefficient (k) and the best responses when the gain (k) is 0.7 (Fig.7.16, Fig.7.17 and Fig.7.18). Tracking curves successfully follow reference curves in a desired time constraint.

Tracking curves are generated by taking feedback values from the encoders attached to the rears of servo motors. Closed loop control tracking curves can be improved by taking feedback values from external encoders attached to the shafts of gears instead of using the encoders of servo motors.

REFERENCES

- Abaqus Inc. (2006). *Abaqus*. Retrieved 2006, from www.abaqus.com.
- Adlink Technology Inc. (2006). *Adlink Technology*. Retrieved 2006, from www.adlinktech.com.
- Alizade, R., & Bayram, C. (2004). Structural Synthesis of Parallel Manipulators. *Mechanism and Machine Theory*, 39, 857-870.
- Bal, G., Bekiroglu, E., Demirbas, S., & Colak, I. (2004). Fuzzy Logic Based DSP Controlled Servo Position Control for Ultrasonic Motor. *Energy Conversion and Management*, 45, 3139-3153.
- Baldor Electric Company. (2007). *Baldor*. Retrieved January 07, 2007, from <http://www.baldor.com/>.
- Crnosija, P., Ajdukovic, S., & Kuzmanovic, B. (1999). Microcomputer Implementation of Optimal Algorithms for Closed-Loop Control of a Hybrid Stepper Motor Drives. *IEEE*, 1, 679-683.
- Custom Solutions Inc. (December 05, 2006). *Custom Solutions*. Retrieved December 22, 2006, from http://www.csi3.com/PC_Stand.htm.
- Dandil, B., Gokbulut, M., & Ata, F. (2004). Doğrusal Olmayan Yük Şartlarındaki Asenkron Motorun YSA-PI Hız Denetimi. *Firat Üniversitesi Fen ve Mühendislik Bilimleri Dergisi*, 16 (1), 39-48.
- Dasgupta, B., & Mruthyunjaya, T.S. (2000). The Stewart Platform Manipulator: A Review. *Mechanism and Machine Theory*, 35, 15-40.

- Dulger, L.C., Kirecci,A., & Topalbekiroglu, M. (2001). AC Servomotorlarının Modellenmesi , Simulasyonu ve Hareket Denetiminde Kullanılması. *10. Ulusal Mak. Teo. Sempozyumu Bil. Kit., 1*, 181-189.
- Gao, F., Li, W., Zhao, X., Jin. Z., & Zhao H. (2002). New Kinematic Structures for 2-, 3-, 4-, and 5- DOF Parallel Manipulator Designs. *Mechanism and Machine Theory, 37*, 1395-1411.
- Gough, V.E., & Whitehall, S.G. (1962). Universal Tyre Test Machine. *Proceedings of the 9th International Technical Congress (FISITA), 1*, 177.
- Grimbleby, J.B. (1995). A Simple Algorithm for Closed Loop Control of Stepping Motors. *IEE Proc.-Electr. Power Appl., 142* (1), 5-13.
- Han, S.H., Kim, Y.H., Ha, I.J., Lee, S.T., & Park, J.J. (1995). A Learning Approach to High Precision Speed Control of Servo Motors. *IEEE, 26* (3), 221-226.
- Hashimoto, H., Yamamoto, H., Yanagisawa, S., & Harashima, F. (1988). Brushless Servo Motor Control Using Variable Structure Approach. *IEEE Transactions on Industry Applications, 24* (1), 160-170.
- Hauge, G.S., & Campbell, M.E. (2004). Sensors and Control of a Space-Based Six-Axis Vibration Isolation System. *Journal of Sound and Vibration, 269*, 913-931.
- Haydon Switch & Instrument (HSI) Corp. (2006). *Haydon Switch & Instrument*. Retrieved 2006, from www.hsi-inc.com.
- Hephaist Seiko Company Ltd. (2006). *Hephaist Seiko*. Retrieved 2006, from <http://www.schaublin.ch/e/index.htm>.
- Hunt, K.H. (1983). Structural Kinematics of In-Parallel-Actuated Robot Arms. *ASME J. Mech. Transm. Autom. Des., 105*, 705-712.

- Ider, S.K. (2005). Inverse Dynamics of Parallel Manipulators in the Presence of Drive Singularities. *Mechanism and Machine Theory*, 40, 33-44.
- Jelenkovic, L., Jakobovic, D., & Budin, L. (2004). Hexapod Structure Evaluation as WEB Service. *Proceedings of the 1st International Conference on Informatics and Robotics (ICINCO)*, 1.
- Karagülle, H., Sarıgül, S., Kıral, Z., Varol, K., & Malgaca, L. (01 July 2006). Mikro-konumlandırıcı Robot Tasarımı ve Prototip İmalatı. *TÜBİTAK Araştırma Projesi 1. Dönem Gelişme Raporu, Project No: 104M373*.
- Karagülle, H., Sarıgül, S., Kıral, Z., Varol, K., & Malgaca, L. (01 January 2007). Mikro-konumlandırıcı Robot Tasarımı ve Prototip İmalatı. *TÜBİTAK Araştırma Projesi 2. Dönem Gelişme Raporu, Project No: 104M373*.
- Khongkoom, N., Kanchanathep, A., Nopnakeepong, S., Tamthong, S., Tunyasirirut, S., & Kagawa, R. (2000). Control of the Position DC Servo Motor by Fuzzy Logic. *IEEE/ASME Transactions on Mechatronics*, 3, 354-357.
- Kim, H.S., Cho, Y.M., & Lee, K. (2005). Robust Nonlinear Task Space Control for 6 DOF Parallel Manipulator. *Automatica*, 41, 1591-1600.
- Ku, S.S., Larsen, G., & Cetinkunt, S. (1998). Fast Tool Servo Control for Ultra-Precision Machining at Extremely Low Rates. *Mechatronics*, 8, 381-393.
- Lenze Company. (2006). *Lenze*. Retrieved 2006, from <http://www.lenze.de/en>.
- Lin, C.L., Jan, H.Y., Hwang, T.S., & Tsai, R.C. (2003). Control Design for a Mixed Rotary and Linear Motors Based Manipulator. *Proceedings of the 2003 IEEE/ASME International Conference on Advanced Intelligent Mechatronics (AIM 2003)*, 1, 1298-1303.

- Lin, F.J., & Chiu, S.L. (1998). Adaptive Fuzzy Sliding-Mode Control for PM Synchronous Servo Motor Drives. *IEE Proc.-Control theory Appl.*, 145 (1), 63-72.
- Lin, F.J., & Wai, R.J. (1998). Hybrid Controller using a Neural Network for a PM Synchronous Servo-Motor Drive. *IEE Proc.-Electr. Power Appl.*, 3, 223-230.
- Lin, Y.C. (1994). The Application of Fuzzy Logic Control to Speed Control of a DC Servo Motor System. *Proceedings of the American Control Conference*, 3, 590-594.
- Lu, C.H. (1997). Design and Implementation of a Digitalized Fuzzy Controller for DC Servo Drives. *IEEE International Conference on Intelligent Processing Systems*, 1, 242-246.
- Maxon Motor Company. (2006). *Maxon Motor*. Retrieved 2006, from www.maxonmotor.com.
- McInroy, J. (1999). Dynamic Modelling of Flexure Jointed Hexapods for Control Purposes. *Proceedings of the 1999 IEEE International Conference on Control Applications*, 1, 508-513.
- McKerrow, P.J. (1991). *Introduction to Robotics*. Sydney: Addison-Wesley Publishing Company.
- Mitutoyo Corporation. (2006). *Mitutoyo Corp*. Retrieved 2006, from <http://www.mitutoyo.co.jp/eng/index.html>.
- Mort, N., Abbod, M.F., & Linkens, D.A. (1995). Comparative Study of Fuzzy DC Servo Motors and Stepper Motors for Mechatronic Systems. *IEE Colloquium on Innovations in Manufacturing Control Through Mechatronics*, 6, 1-5.

- MSC Software Corporation. (2006). *MSC Software*. Retrieved 2006, from www.mssoftware.com.
- MSDN. (2007). *Microsoft Corporation*. Retrieved January 14, 2007, from <http://msdn2.microsoft.com/en-us/vbasic/default.aspx>.
- Noorani, R.I. (1990). Microcomputer-Based Robot Arm Control. *Mathematical and Computer Modelling*, 14, 450-455.
- Omron Corporation. (2006). *Omron*. Retrieved 2006, from www.omron.com.
- Physik Instrumente (PI) Company. (2006). *PI*. Retrieved 2006, from www.pi.com.
- RS Company. (2006). *RS Company Turkey*. Retrieved 2006, from www.rsturkey.com.
- Schaublin SA Corporation. (2004). *Schaublin SA*. Retrieved 2006, from <http://www.schaublin.ch/e/index.htm>.
- SolidWorks Corporation. (2007). *SolidWorks*. Retrieved January 14, 2007, from www.solidworks.com.
- Stewart, D. (1965). A Platform with Six Degrees of Freedom. *Proceedings of the Institution of Mechanical Engineers*, 180, 371-386.
- Tsai, L.W. (1999). *Robot Analysis (The Mechanics of Serial and Parallel Manipulators)*. New York: John Wiley & Sons Inc.
- Tzou, Y.Y., Wu, H.J. (1990). Multimicroprocessor-Based Robust Control of an AC Induction Servo Motor. *IEEE Transactions on Industry Applications*, 26 (3), 441-449.

- Van de Straete, H.J., Degezelle, P., De Schutter, J., & Belmans, R.J.M. (1998). Servo Motor Selection Criterion for Mechatronic Applications. *IEEE/ASME Transactions on Mechatronics*, 3 (1), 43-50.
- Wendlandt, J.M., & Sastry, S.S. (1994). Design and Control of a Simplified Stewart Platform for Endoscopy. *Proceedings of the 33rd IEEE Conference on Decision and Control*, 1, 357-362.
- Ximei, Z., & Qingding, G. (2005). H_∞ Robust Control Based on International Model Theory for Linear Permanent Magnet Synchronous Motor. *Proceedings of the Eighth International Conference on Electrical Machines and Systems (ICEMS 2005)*, 2, 1613-1616.
- Yamamoto, K., & Shinohara, K. (1996). Comparison Between Space Vector Modulation and Subharmonic Methods for Current Harmonics of DSP - Based Permanent - Magnet AC Servo Motor Drive System. *IEE Proc.-Electr. Power Appl.*, 143 (2), 151-156.
- Yang, M.Y., & Hong, W.P. (2001). A PC-NC Milling Machine with New Simultaneous 3-Axis Control Algorithm. *International Journal of Machine Tools & Manufacture*, 41, 555-566.
- Yaskawa. (2002). *Σ -II Series SGMBH/SGDH User's Manual*. Tokyo: Yaskawa Electric Corporation.
- Yoneya, A., Yoshimaru, K., & Togari, Y. (2000). Self-Sensing Control of AC-Servo Motor with DSP Oriented Observer. *IEEE*, 1, 560-565.

APPENDIX A
PROPERTIES OF ADLINK PCI 8132 AND PCI 8164 MOTION CONTROL
CARDS

A.1 Features of PCI 8132 and PCI 8164

The PCI 8132 and PCI 8164 cards are 2 axes and 4 axes motion control cards with PCI interface, respectively. It can generate high frequency pulse trains to drive stepping motors and servo motors. Multiple PCI 8132 and PCI 8164 cards can be used in one system. Incremental encoder interface provide the ability to correct for positioning errors generated by inaccurate mechanical transmissions (Adlink Inc., 2006).

The following lists summarize the main features of the PCI 8132 motion control card. The information listed below can be found at Adlink Co. (Adlink Inc., 2006).

- 32-bit PCI bus, plug and play.
- 2 axes of step and direction pulse output for controlling stepping or servomotor.
- Maximum pulse output frequency: 2.4Mpps, linear, trapezoidal or S curve velocity profile drive.
- 2 axes circular and linear interpolation.
- 0~268.435.455 or -134.217.728 to +134.217.727, 28-bit up/down counter for incremental encoder feedback.
- Home switch, index signal, positive and negative limit switches interface provided for all axes.
- Programmable interrupt sources.
- Change speed on the fly.
- Position compare and trigger signal output.
- Simultaneous start/stop motion on multiple axes.
- Manual pulser input interface.
- Software supports maximum up to 12 PCI 8132 cards (24 axes) operation.

- PCI 8132 library and utility for DOS library and Windows 95/98/NT DLL.
- Internal reference clock: 9.8304 MHz.
- Pulse rate setting steps: 0 to 2.4Mpps.
- Position comparison range:-8,388,608 ~ +8388607 (24 bit).
- Position pulse setting range: 0~268,435,455 pulses (28-bit).

The following lists summarize the main features of the PCI 8164 motion control card. The information listed below can be found at Adlink Co. (Adlink Inc., 2006).

- 32-bit PCI bus, plug and play.
- 4 axes of step and direction pulse output for controlling stepping or servomotor.
- 6.55MPPS maximum pulse output frequency, linear, trapezoidal, or S-Curve velocity profile drive.
- Any 2 of 4 axes circular interpolation.
- Any 2-4 of 4 axes linear interpolation.
- Continuous interpolation for contour following motion.
- Change position and speed on the fly.
- Change speed by condition comparing.
- 13 home return modes with auto searching.
- Hardware backlash compensator and vibration suppression.
- 2 software end-limits for each axis.
- 0~268.435.455 or -134.217.728 to +134.217.727, 28-bit up/down counter for incremental encoder feedback.
- 2-axis high speed position latch input.
- 2-axis position compare trigger output with 4k FIFO auto loading.
- Simultaneous start/stop motion on multiple axes.
- Manual pulser input interface.
- Software supports a maximum of up to 12 PCI-8164 cards (48 axes) operation in one system.
- Libraries and utilities support DOS, Windows® 9X/NT/2000/XP, and Linux.

- 19.66 MHz internal reference clock.
- Pulse rate setting ranges (pulse ratio = 1: 65535).
- Position pulse setting range (28-bit): -134,217,728 to +134,217,728.

A.2 Fundamental Commands of PCI 8132 and PCI 8164

Fundamental commands of PCI motion control cards used in the programs with respect of VisualBASIC are presented in Table A.1. Suitable commands which are related to desired motion profile and feedback type, can be selected from the Table A.1.

Table A.1 Fundamental commands of the motion control cards

Commands for PCI 8132	Commands for PCI 8164	Explanation
B_8132_set_pls_outmode	B_8164.Axis0.OutputMode	Type of the motion.
B_8132_set_cnt_src	B_8164_set_feedback_src	External encoder (0 means external encoder, 1 means internal encoder; they are opposite for PCI 8132).
B_8132_get_position	B_8164_get_position	Reads feedback.
B_8132_start_t_move	B_8164_tv_move	Trapezoidal constant motion.
B_8132_start_s_move	B_8164_sv_move	S profile constant motion.
B_8132_v_stop	B_8164_sd_stop	Stops the motion.
B_8132_start_ts_move	B_8164_start_tr_move	Trapezoidal relative motion.
B_8132_start_ta_move	B_8164_start_ta_move	Trapezoidal absolute motion.
B_8132_start_rs_move	B_8164_start_sr_move	S profile relative motion.
B_8132_start_ra_move	B_8164_start_sa_move	S profile absolute motion.
B_8132_start_move_xy	B_8164_start_tr_line2 (3,4) B_8164_start_ta_line2 (3,4) B_8164_start_sr_line2 (3,4) B_8164_start_sa_line2 (3,4)	Linear interpolation with respect to motion profile and desired 2 (or 3 or 4) axes.
B_8132_arc_xy	B_8164_start_a_arc2 B_8164_start_r_arc2	Circular interpolation with respect to motion profile and desired 2 axes.
B_8132_v_change	B_8164_v_change	Changes speed on fly.

APPENDIX B
ACCELERATION – DECELERATION SINUSOID

B.1 Definition of the Acceleration – Deceleration Sinusoid

Velocity – time graph for velocity profiles which are used as velocity inputs is given in Figure B.1 (Karagülle et al., 2006).

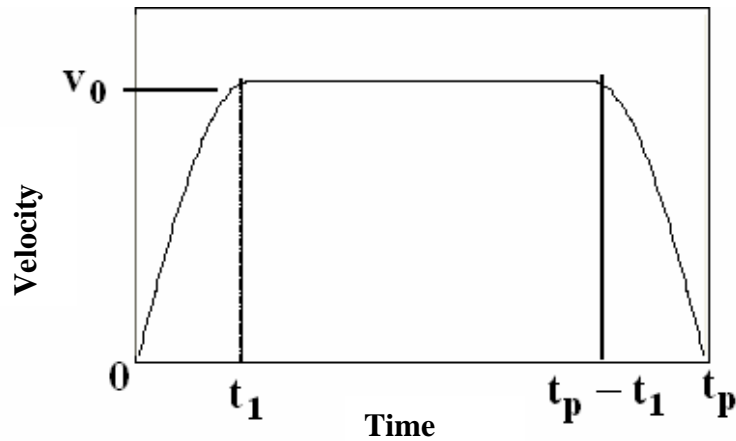


Figure B.1 Velocity – time graph for velocity inputs.

Duration of the motion is t_p and there is acceleration in $0 - t_1$ time interval and deceleration between $t_p - t_1$ time interval and t_p . Constant velocity which is between t_1 and $t_p - t_1$, is V_0 . Acceleration and deceleration curves are a quarter sinus curves. Suppose that the value of whatever positions or angular degrees of freedom is S_A at $t = 0$ and S_B at $t = t_p$. V_0 is found by equalling $S_B - S_A$ to the area under the velocity curve.

B.2 Creating Samples of the Sinusoid by VisualBASIC

A developed VisualBASIC subroutine which finds samples of velocity values whose inputs are S_A , S_B , t_p , N_s and N_l is presented below. If Δt is sampling period, $N_s (\Delta t) = t_p$ and $N_l (\Delta t) = t_1$ (Karagülle et al., 2006).

```

Rem: Inputs: sa, sb, tp, ns, n1 Results: ts(k), sv(k)
dt = tp / ns: t1 = n1 * dt: w = pi / (2 * t1): t2 = tp - t1
v0 = (sb - sa) / (2 / w + (t2 - t1)): t = 0
For k = 0 To n1
    sv(k) = v0 * Sin(w * t): ts(k) = t: t = t + dt
Next k
For k = n1 + 1 To ns - n1 - 1
    sv(k) = v0: ts(k) = t: t = t + dt
Next k
For k = ns - n1 To ns
    sv(k) = v0 * Cos(w * (t - t2)): ts(k) = t: t = t + dt
Next k
Return

```

APPENDIX C
2D MANUFACTURING DRAWINGS OF THE HEXAPOD

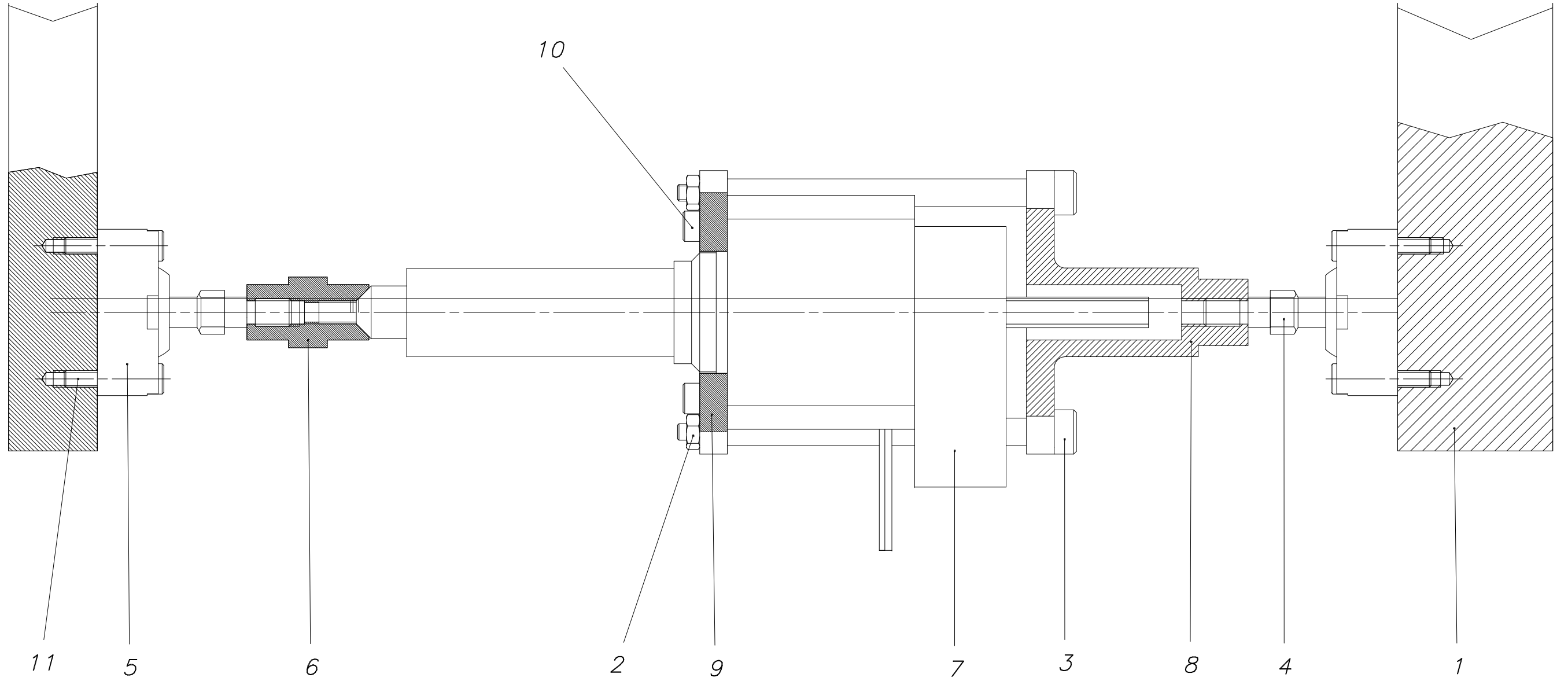
In this appendix, 2D drawings of parts, which are manufactured, of the hexapod is presented. The drawings are generated from 3D solid models of the hexapod in I-Deas solid modelling program (Karagülle et al., 2007).

The assembly of one axis is shown at Page 94. Part names which are related to part numbers of the assembly are given in Table C.1.

Table C.1 Identification of the assembly numbers.

Number	Item	Full name	Name
1	1	Lower platform	pa
2	4	M3 nut of stud bolt	spn
3	4	M3 stud bolt	sp
4	1	Shaft of spherical joint	lsa
5	1	Spherical joint	sa
6	1	Joint connection part	lcb
7	1	Linear motor	lma
8	1	Lower connection part	lca
9	1	Upper connection part	lda
10	4	Capscrews	M3x10
11	4	Capscrews	M3x15
12	1	Upper platform	pb

2D drawings of the lower platform and the upper platform are presented at Page 95 and Page 96, respectively. The joint connection part is at Page 97. The lower connection part and the upper connection part of the motor are given at Page 98 and Page 99, respectively. The part named stud bolt, which connects the lower connection part and the upper connection part, is shown at Page 100.

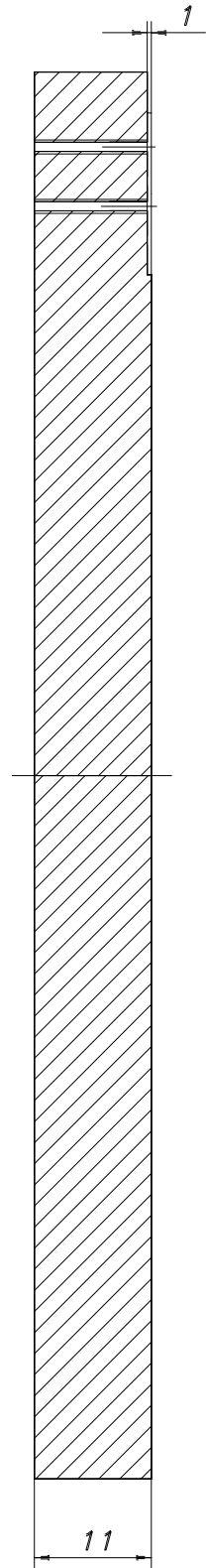
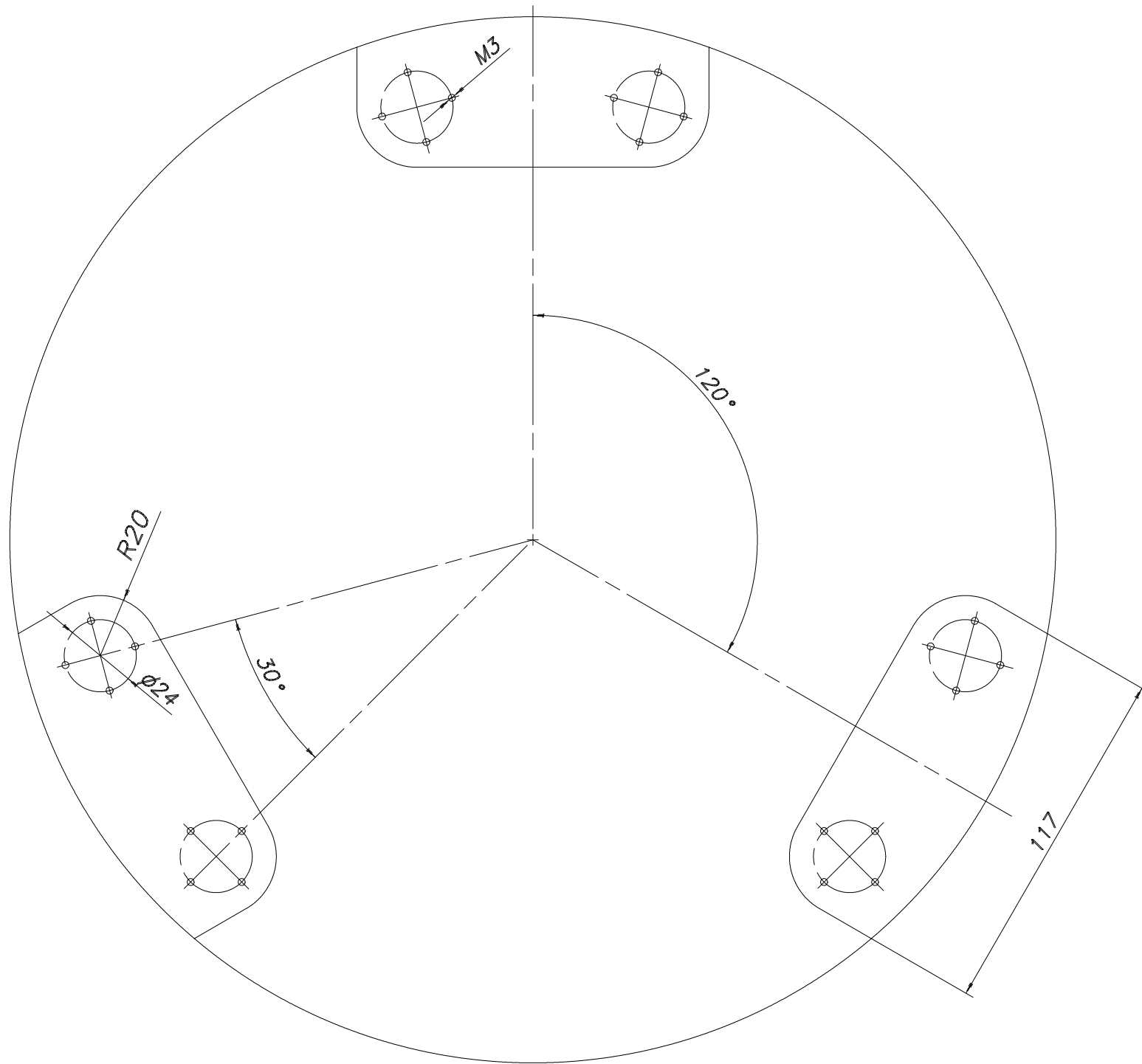


No	Adet	Parca adi	Cizim No	Malzeme
12	1	ust tabla	HEX_00_12	
11	4	Silindirik civata	M3x15	
10	4	silindirik civata	M3x10	
9	1	Motor ust baglanti plakasi	HEX_00_02	
8	1	Motor alt baglanti plakasi	HEX_00_04	
7	1	Motor	HEX_00_01	
6	1	Mafsal Motor baglantisi	HEX_00_09	
5	1	Mafsal baglanti bloğu	HEX_00_07	
4	1	küresel mafsal mili	HEX_00_10	
3	4	Civata M3	HEX_00_05	
2	4	Altikose somun M3	HEX_00_06	
1	1	alt tabla	HEX_00_13	

BELİRTİLMEYEN TOLERANSLAR		ONAYLAR		TARİH	
ÖLÇÜLER MİLMİTREDİR		CİZEN		16.07.06	
1 - 6 ±0,1		KONTROL		ONAY	
6 - 30 ±0,2		TARİH		16.07.06	
30 - 100 ±0,3		CİZİM ADI		hexapod.mf1	
1000 - 2000 ±1,2		MALZEME		HEXAPOD TEK EKSEN ELEMANLARI	
AÇILAR: 0 - 30'		CAD DOSYA NO.		HEX_00_01	
CİZİMDEN ÖLÇÜ ALMAYINIZ		CİZİM NO.		HEX_00_01	
DEĞİŞİKLİKLER		OLÇEK		2:1	
ONAY		REV. NO.		SAYFA NO.	

B B A T Ü L
BİLGİSAYARLA ANALİZ
TASARIM ÜRETİM
LABORATUVARI

DOKUZ EYLÜL ÜNİVERSİTESİ MÜHENDİSLİK FAKÜLTESİ
MAKİNA MÜHENDİSLİĞİ BÖLÜMÜ



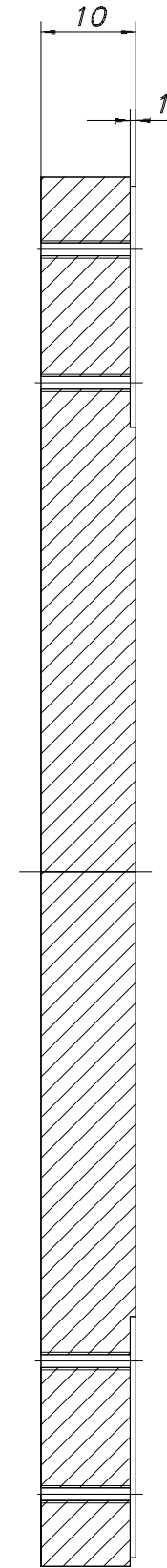
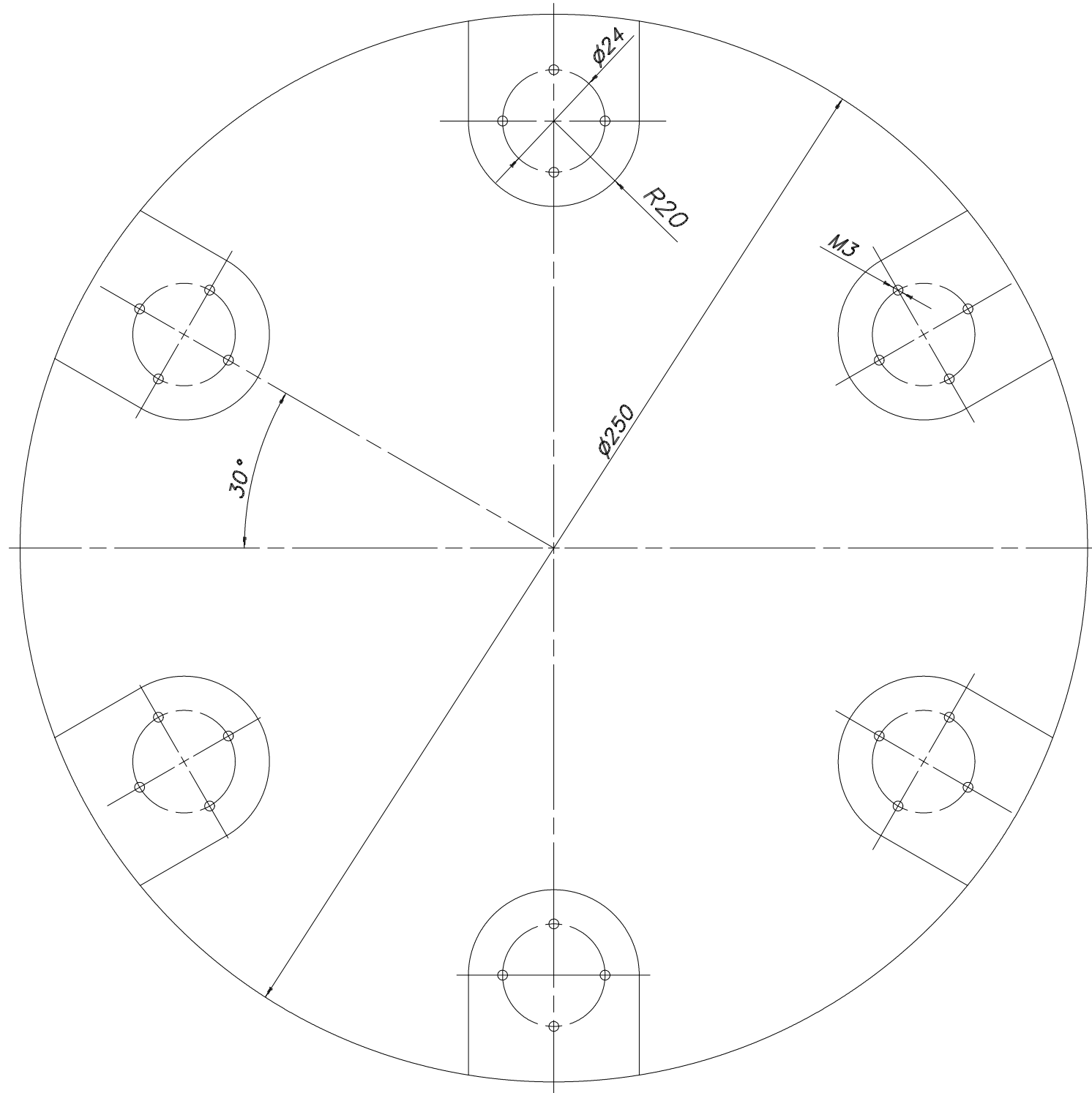
BELİRTİLMİYEN TOLERANSLAR
ÖLÇÜLER MİLMİTREDİR

ÖLÇÜLER			
1 - 6	±0.1	100 - 300	±0.5
6 - 30	±0.2	300 - 1000	±0.8
30 - 100	±0.3	1000 - 2000	±1.2

AÇILAR: 0 - 30°
ÇİZİMDEN ÖLÇÜ ALMAYINIZ
DEĞİŞİKLİKLER

B B A T Ü L BİLGİSAYARLA ANALİZ TASARIM ÜRETİM LABORATUVARI	
ONAYLAR	TARİH
ÇİZEN <i>Kemal Varol</i>	16.07.06
KONTROL	
ONAY	

DOKUZ EYLÜL ÜNİVERSİTESİ MÜHENDİSLİK FAKÜLTESİ MAKİNA MÜHENDİSLİĞİ BÖLÜMÜ			
MALZEME	ÇİZİM ADI ALT TABLA		
CAD DOSYA NO.	ÇİZİM NO. HEX-00-13		
ÖLÇEK 1:1	REV. NO.	SAYFA NO.	



BELİRTİLMİYEN TOLERANSLAR
ÖLÇÜLER MİLMİTREDİR

ÖLÇÜLER			
1 - 6	±0.1	100 - 300	±0.5
6 - 30	±0.2	300 - 1000	±0.8
30 - 100	±0.3	1000 - 2000	±1.2

AÇILAR: 0 - 30'
ÇİZİMDEN ÖLÇÜ ALMAYINIZ
DEĞİŞİKLİKLER

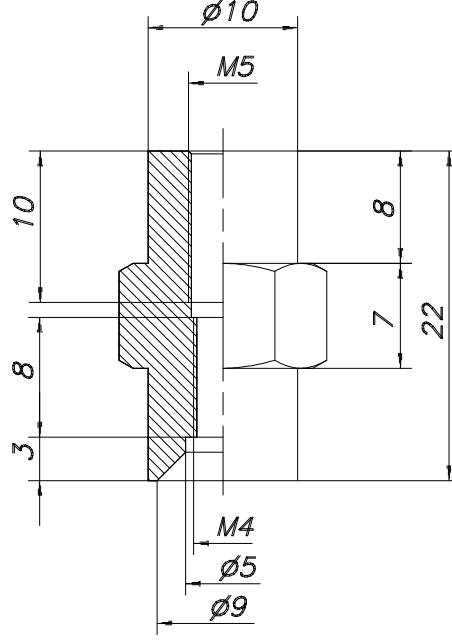
ONAYLAR		TARİH	
		ÇİZEN	Kemal Varol
KONTROL			
ONAY			

DOKUZ EYLÜL ÜNİVERSİTESİ MÜHENDİSLİK FAKÜLTESİ MAKİNA MÜHENDİSLİĞİ BÖLÜMÜ			
MALZEME		ÇİZİM ADI	
		ÜST TABLA	
CAD DOSYA NO.		ÇİZİM NO.	
		HEX-00-12	
ÖLÇEK	1:1	REV NO.	SAYFA NO.
			1

4 3 2 1

D

D

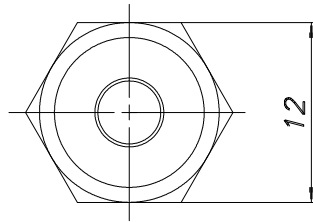


C

C

B

B



A

A

BELİRTİLMİYEN TOLERANSLAR
ÖLÇÜLER MİLMETREDİR

ÖLÇÜLER	±0.1	100 - 300	±0.5
1 - 6	±0.2	300 - 1000	±0.8
6 - 30	±0.3	1000 - 2000	±1.2
30 - 100			

ACILAR: 0 - 30°

ÇİZİMDEN ÖLÇÜ ALMAYINIZ

DEĞİŞİKLİKLER

ONAY TARİH



B B A T Ü L
BİLGİSAYARLI ANALİZ
TASARIM ÜRETİM
LABORATUVARI

ONAYLAR		TARİH
ÇİZEN	<i>Kemal Varol</i>	<i>16.07.06</i>
KONTROL		
ONAY		

DOKUZ EYLÜL ÜNİVERSİTESİ MÜHENDİSLİK FAKÜLTESİ
MAKİNA MÜHENDİSLİĞİ BÖLÜMÜ

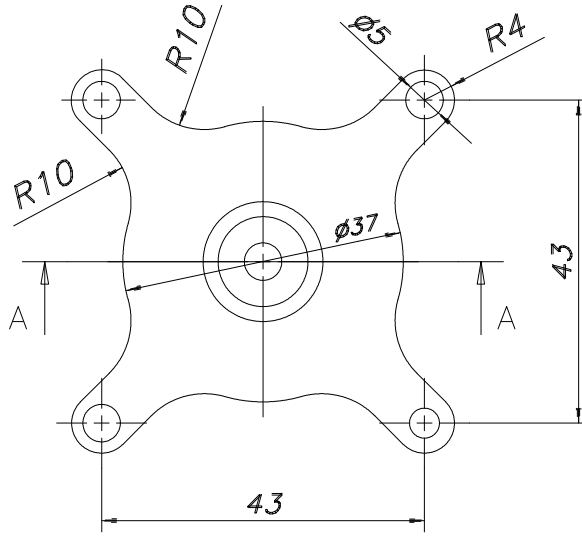
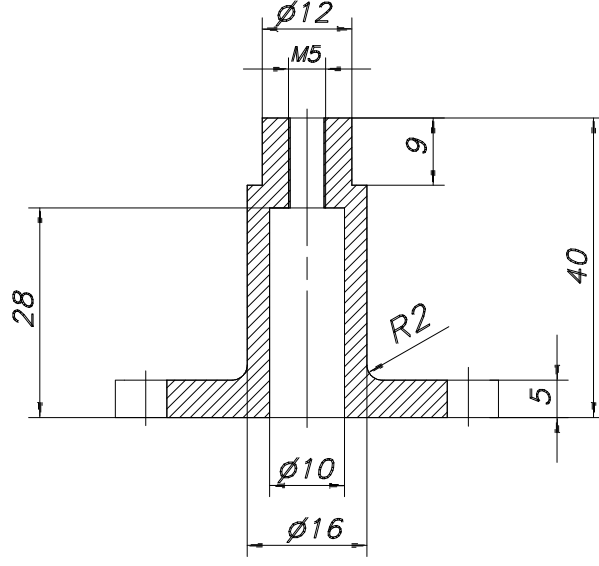
MALZEME	ÇİZİM ADI <i>MOTOR-MAFSAL BAĞLANTI ELEMANI</i>
CAD DOSYA NO.	ÇİZİM NO. <i>HEX-00-09</i>
ÖLÇEK <i>2:1</i>	REV NO.
	SAYFA NO.

4

3

2

1



BELİRTİLMİYEN TOLERANSLAR
ÖLÇÜLER MİLMETREDİR

ÖLÇÜLER			
1 - 6	±0.1	100 - 300	±0.5
6 - 30	±0.2	300 - 1000	±0.8
30 - 100	±0.3	1000 - 2000	±1.2

ACILAR: 0 - 30°

ÇİZİMDEN ÖLÇÜ ALMAYINIZ

DEĞİŞİKLİKLER

ONAY TARİH

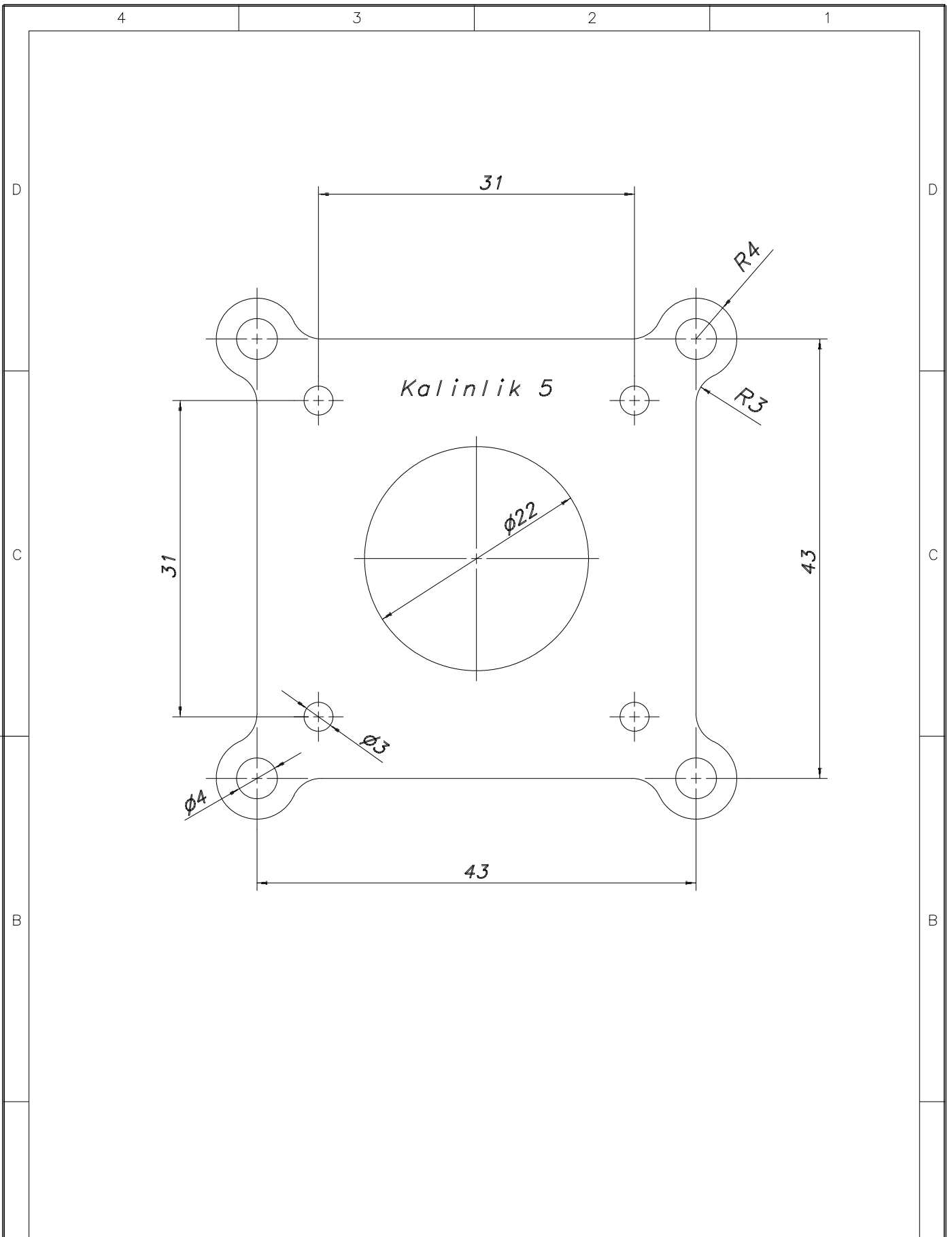


B B A T Ü L
BİLGİSAYARLA ANALİZ
TASARIM ÜRETİM
LABORATUVARI

ONAYLAR		TARİH
ÇİZEN	<i>Kemal Varol</i>	16.07.06
KONTROL		
ONAY		

DOKUZ EYLÜL ÜNİVERSİTESİ MÜHENDİSLİK FAKÜLTESİ
MAKİNA MÜHENDİSLİĞİ BÖLÜMÜ

MALZEME	ÇİZİM ADI MOTOR ALT BAĞLANTI PLAKASI
CAD DOSYA NO.	ÇİZİM NO. HEX-00-04
ÖLÇEK 1:1	REV NO.
	SAYFA NO.



BELİRTİLMİYEN TOLERANSLAR
ÖLÇÜLER MİLMETREDİR

ÖLÇÜLER			
1 - 6	±0.1	100 - 300	±0.5
6 - 30	±0.2	300 - 1000	±0.8
30 - 100	±0.3	1000 - 2000	±1.2

AÇILAR: 0 - 30°

ÇİZİMDEN ÖLÇÜ ALMAYINIZ

DEĞİŞİKLİKLER

ONAY TARİH

B B A T Ü L
BİLGİSAYARLA ANALİZ
TASARIM ÜRETİM
LABORATUVARI

ONAYLAR		TARİH
ÇİZEN	<i>Kemal Varol</i>	<i>16.07.06</i>
KONTROL		
ONAY		

DOKUZ EYLÜL ÜNİVERSİTESİ MÜHENDİSLİK FAKÜLTESİ
MAKİNA MÜHENDİSLİĞİ BÖLÜMÜ

MALZEME	ÇİZİM ADI <i>MOTOR ÜST BAĞLANTI PLAKASI</i>
CAD DOSYA NO.	ÇİZİM NO. <i>HEX-00-02</i>
ÖLÇEK <i>2:1</i>	REV NO.
	SAYFA NO.

4

3

2

1

D

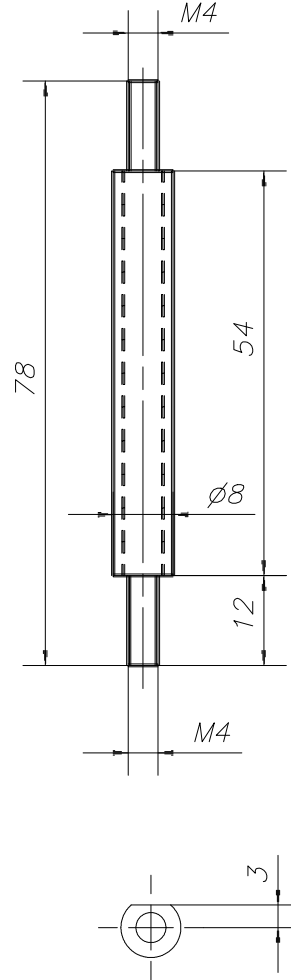
D

C

C

B

B



A

A

BELİRTİLMİYEN TOLERANSLAR
ÖLÇÜLER MİLMETREDİR

ÖLÇÜLER			
1 - 6	±0.1	100 - 300	±0.5
6 - 30	±0.2	300 - 1000	±0.8
30 - 100	±0.3	1000 - 2000	±1.2

ACILAR: 0 - 30°

ÇİZİMDEN ÖLÇÜ ALMAYINIZ

DEĞİŞİKLİKLER

ONAY

TARİH



B A T Ü L
BİLGİSAYARLA ANALİZ
TASARIM ÜRETİM
LABORATUVARI

DOKUZ EYLÜL ÜNİVERSİTESİ MÜHENDİSLİK FAKÜLTESİ
MAKİNA MÜHENDİSLİĞİ BÖLÜMÜ

MALZEME

ÇİZİM ADI

SAPLAMA

CAD DOSYA NO.

ÇİZİM NO.

ÖLÇEK

REV NO.

SAYFA NO.

ONAYLAR

TARİH

ÇİZEN

K. Varol

09.06

KONTROL

ONAY

4

3

2

1

APPENDIX D
VISUALBASIC PROGRAM FOR SIMULATION AND CONTROLLING
OF THE HEXAPOD

```
Private Sub Command1_Click()
```

```
    Call assemble1
```

```
End Sub
```

```
Private Sub Command10_Click()
```

```
    v0puls = 3000: t1 = 0.01
```

```
    dis = -dis0 / 0.00075: Call move_motors
```

```
End Sub
```

```
Private Sub Command2_Click()
```

```
    Call forward1
```

```
End Sub
```

```
Private Sub Command3_Click()
```

```
    Call inverse1
```

```
End Sub
```

```
Private Sub Command4_Click()
```

```
    Call rmeter1
```

```
End Sub
```

```
Private Sub Command6_Click()
```

```
    Call move1
```

```
End Sub
```

```
Private Sub Command7_Click()
```

```
    End
```

```
End Sub
```

```
Private Sub Command8_Click()
```

```
    xc = InputBox("Motor number:"); If xc = "" Then Exit Sub
```

```
    nmotor = Val(xc): Text1.Text = Str(nmotor) + "," + Str(dis0)
```

```
End Sub
```

```

Private Sub Command9_Click()
    xc = InputBox("Distance:"); If xc = "" Then Exit Sub
    dis0 = Val(xc): Text1.Text = Str(nmotor) + "," + Str(dis0)
End Sub

```

```

Private Sub Form_Activate()
    Call adlink0
End Sub

```

```

Private Sub Form_Load()
    nmotor = 1: dis0 = 0.5: pi = 4 * Atn(1)
    Form1.Caption = "Hexapod1"
    WindowState = 2: Form1.AutoRedraw = True
    Text1.Text = ""
    Command1.Caption = "Assemble"
    Command3.Caption = "Motion-inverse"
    Command4.Caption = "Actuator lengths"
    Command2.Caption = "Motion-forward"
    Command6.Caption = "Move"
    Command10.Caption = "move_0"
    Command8.Caption = "motor"
    Command9.Caption = "dis0"
    Command7.Caption = "end"
End Sub

```

-----Assemble.bas-----

```

Public Const fl0 = "d:\hexapod1\"
Public doc As Object, pi As Double
Public xsa(6), ysa(6), xsb(6), ysb(6) As Double
Public konwm As Integer
Public pad1, pat1, par2, pafi1 As Double
Public pbd1, pbt1, pbr2, pbfi1 As Double
Public heph_c, h1, l1, r0 As Double
Public ns, nframe As Integer
Public dt As Double
Public zsa, zsb As Double
Sub const0()
    pi = Atn(1) * 4

```

```

pad1 = 0.011: pat1 = 0.001: par2 = 0.174 - 0.025: pafi1 = 30 * pi / 180
pbd1 = 0.01: pbt1 = pat1: pbr2 = 0.125 - 0.025: pbfi1 = 60 * pi / 180
heph_c = 0.007: h1 = 244.1362 / 1000
ns = 20: nframe = 5: dt = 0.001

```

```
'---
```

```

Call heph1: x = xsb(1) * 1000 - xsa(1) * 1000: y = ysb(1) * 1000 - ysa(1) * 1000
zsa = 0.5 * pad1 - pat1 + heph_c: zsb = -0.5 * pbd1 + pbt1 - heph_c
z = (h1 - 0.5 * pad1 - 0.5 * pbd1 - zsa + zsb) * 1000
l1 = Sqr(x ^ 2 + y ^ 2 + z ^ 2): r0 = l1 - (0.18 + 0.01) * 1000
pbz0 = (h1 - 0.5 * pad1 - 0.5 * pbd1) * 1000

```

```
End Sub
```

```

Sub assemble1()           'Assembles parts
  xc = InputBox("Continue", , "y"): If xc <> "y" Then Exit Sub
  Call wm3d0: If doc.Name <> "hexapod1.WM3" Then Exit Sub
  Call heph2
  Call name1
  Call align1
  Call cmeter1

```

```
End Sub
```

```

Sub align1()             'Align parts
  z = h1 - 0.5 * pad1 - 0.5 * pbd1 - zsa + zsb
  cj = "1": GoSub 60
  cj = "2": GoSub 60
  cj = "3": GoSub 60
  cj = "4": GoSub 60
  cj = "5": GoSub 60
  cj = "6": GoSub 60
  Exit Sub

```

```

60 b1c = "lsa" + cj: b2c = "lsb" + cj: kj = Val(cj)
x = xsb(kj) - xsa(kj): y = ysb(kj) - ysa(kj)
rx = Atn(-y / z): cos2 = Sqr(y * y + z * z) / l1: sin2 = x / l1: ry = Atn(sin2 / cos2)
a = doc.Bodies(b1c).SetConfig(xsa(kj), ysa(kj), 0.5 * h1, rx, ry, 0)
a = doc.Bodies(b2c).SetConfig(xsa(kj), ysa(kj), 0.5 * h1, rx, ry, 0)
c1c = "sa-s" + cj: c2c = "lsa-s" + cj
Set c1 = doc.Coords(c1c): Set c2 = doc.Coords(c2c)
Call doc.Constraints.CreateConstraint(3, 3, c2, c1)

```



```

c1c = "sb-s" + cj: c2c = "lsb-s" + cj
Set c1 = doc.Coords(c1c): Set c2 = doc.Coords(c2c)
Call doc.Constraints.CreateConstraint(3, 3, c2, c1)
c1c = "lma-l" + cj: c2c = "lmb-l" + cj
Set c1 = doc.Coords(c1c): Set c2 = doc.Coords(c2c)
Call doc.Constraints.CreateConstraint(4, 1, c2, c1)
Call doc.Constraints.CreateConstraint(7, 1, c2, c1)
nc = doc.Constraints.Count: doc.Constraints(nc).Name = "la" + cj
doc.Constraints("la" + cj).ActuatorValue.value = 0
Return

```

End Sub

```

Sub cmeter1()           'Constraint meters
  cj = "la1": cjj = "-l": GoSub 50
  doc.Meters(n).Column(1).Formula = "constraint[" & (doc.Constraints("la1").ID) & "].length"
  cj = "la2": GoSub 50
  doc.Meters(n).Column(1).Formula = "constraint[" & (doc.Constraints("la2").ID) & "].length"
  cj = "la3": GoSub 50
  doc.Meters(n).Column(1).Formula = "constraint[" & (doc.Constraints("la3").ID) & "].length"
  cj = "la4": GoSub 50
  doc.Meters(n).Column(1).Formula = "constraint[" & (doc.Constraints("la4").ID) & "].length"
  cj = "la5": GoSub 50
  doc.Meters(n).Column(1).Formula = "constraint[" & (doc.Constraints("la5").ID) & "].length"
  cj = "la6": GoSub 50
  doc.Meters(n).Column(1).Formula = "constraint[" & (doc.Constraints("la6").ID) & "].length"
  '---
Exit Sub
50 a = doc.Meters.Add(): n = doc.Meters.Count: doc.Meters(n).Name = cj + cjj
doc.Meters(n).AllocateColumns (2): doc.Meters(n).Visible = False
Return

```

End Sub

```

Sub name1()           'Renames
  doc.Bodies("Copy of lsa1").Name = "lsa2"
  doc.Bodies("Copy (2) of lsa1").Name = "lsa3"
  doc.Bodies("Copy (3) of lsa1").Name = "lsa4"
  doc.Bodies("Copy (4) of lsa1").Name = "lsa5"
  doc.Bodies("Copy (5) of lsa1").Name = "lsa6"
  '---

```

```

doc.Bodies("Copy of lsb1").Name = "lsb2"
doc.Bodies("Copy (2) of lsb1").Name = "lsb3"
doc.Bodies("Copy (3) of lsb1").Name = "lsb4"
doc.Bodies("Copy (4) of lsb1").Name = "lsb5"
doc.Bodies("Copy (5) of lsb1").Name = "lsb6"
'---
doc.Coords("Copy of lsa-s1").Name = "lsa-s2"
doc.Coords("Copy (2) of lsa-s1").Name = "lsa-s3"
doc.Coords("Copy (3) of lsa-s1").Name = "lsa-s4"
doc.Coords("Copy (4) of lsa-s1").Name = "lsa-s5"
doc.Coords("Copy (5) of lsa-s1").Name = "lsa-s6"
'---
doc.Coords("Copy of lsb-s1").Name = "lsb-s2"
doc.Coords("Copy (2) of lsb-s1").Name = "lsb-s3"
doc.Coords("Copy (3) of lsb-s1").Name = "lsb-s4"
doc.Coords("Copy (4) of lsb-s1").Name = "lsb-s5"
doc.Coords("Copy (5) of lsb-s1").Name = "lsb-s6"
'---
doc.Coords("Copy of lma-11").Name = "lma-12"
doc.Coords("Copy (2) of lma-11").Name = "lma-13"
doc.Coords("Copy (3) of lma-11").Name = "lma-14"
doc.Coords("Copy (4) of lma-11").Name = "lma-15"
doc.Coords("Copy (5) of lma-11").Name = "lma-16"
'---
doc.Coords("Copy of lmb-11").Name = "lmb-12"
doc.Coords("Copy (2) of lmb-11").Name = "lmb-13"
doc.Coords("Copy (3) of lmb-11").Name = "lmb-14"
doc.Coords("Copy (4) of lmb-11").Name = "lmb-15"
doc.Coords("Copy (5) of lmb-11").Name = "lmb-16"
'---

```

End Sub

```

Sub heph1()                                'Parameters
bc = "pa": R = par2
fi = 0.5 * pafi1: k = 1: GoSub 100
fi = 120 * pi / 180 - 0.5 * pafi1: c1c = "sa-s2": k = 2: GoSub 100
fi = 120 * pi / 180 + 0.5 * pafi1: c1c = "sa-s3": k = 3: GoSub 100
fi = 240 * pi / 180 - 0.5 * pafi1: c1c = "sa-s4": k = 4: GoSub 100
fi = 240 * pi / 180 + 0.5 * pafi1: c1c = "sa-s5": k = 5: GoSub 100

```

```

fi = -0.5 * pafi1: k = 6: GoSub 100
'---
bc = "pb": R = pbr2
fi = 0.5 * pbf1: c1c = "sb-s1": k = 1: GoSub 100
fi = 120 * pi / 180 - 0.5 * pbf1: c1c = "sb-s2": k = 2: GoSub 100
fi = 120 * pi / 180 + 0.5 * pbf1: c1c = "sb-s3": k = 3: GoSub 100
fi = 240 * pi / 180 - 0.5 * pbf1: c1c = "sb-s4": k = 4: GoSub 100
fi = 240 * pi / 180 + 0.5 * pbf1: c1c = "sb-s5": k = 5: GoSub 100
fi = -0.5 * pbf1: c1c = "sb-s6": k = 6: GoSub 100
Exit Sub
'---
100 x = R * Cos(fi): y = R * Sin(fi)
If bc = "pa" Then xsa(k) = x: ysa(k) = y
If bc = "pb" Then xsb(k) = x: ysb(k) = y
Return
End Sub

Sub heph2()                'Parameters
z = h1 - 0.5 * pad1 - 0.5 * pbd1: a = doc.Bodies("pb").SetConfig(0, 0, z, 0, 0, 0)
'---
bc = "pa": z = 0.5 * pad1 - pat1 + heph_c
c1c = "sa-s1": x = xsa(1): y = ysa(1): GoSub 100
c1c = "sa-s2": x = xsa(2): y = ysa(2): GoSub 100
c1c = "sa-s3": x = xsa(3): y = ysa(3): GoSub 100
c1c = "sa-s4": x = xsa(4): y = ysa(4): GoSub 100
c1c = "sa-s5": x = xsa(5): y = ysa(5): GoSub 100
c1c = "sa-s6": x = xsa(6): y = ysa(6): GoSub 100
'---
bc = "pb": R = pbr2: z = -0.5 * pbd1 + pbt1 - heph_c
c1c = "sb-s1": x = xsb(1): y = ysb(1): GoSub 100
c1c = "sb-s2": x = xsb(2): y = ysb(2): GoSub 100
c1c = "sb-s3": x = xsb(3): y = ysb(3): GoSub 100
c1c = "sb-s4": x = xsb(4): y = ysb(4): GoSub 100
c1c = "sb-s5": x = xsb(5): y = ysb(5): GoSub 100
c1c = "sb-s6": x = xsb(6): y = ysb(6): GoSub 100
Exit Sub
'---
100 doc.Coords.Add: n = doc.Coords.Count
Set doc.Coords(n).Body = doc.Bodies(bc): doc.Coords(n).Name = c1c

```

```

a = doc.Coords(n).SetConfig(x, y, z, 0, 0, 0)
Return
End Sub

Sub wm3d0()           'Initialize
  If konwm > 0 Then Exit Sub
  Set wm = GetObject("WM3D.Application"): Set doc = wm.ActiveDocument
  Call const0: konwm = 1
  doc.AccuracySettings.IntegrationStepsPerFrame = nframe
  doc.AccuracySettings.IntegrationSecondsPerStep = dt / nframe
  doc.RunControl.ActionType = 1
  doc.RunControl.Condition.Formula = "frame()=" + Str(ns)
End Sub

```

-----**Simulate.bas**-----

```

Public pbz0 As Double, kerror As Integer
Public pbx As Double, pby As Double, pbz As Double
Public pbrx As Double, pbry As Double, pbrz As Double
Public ra(6) As Double, la(6) As Double
Public thc As String, th As Double

Sub inverse1()       'Inverse kinematics
  Call m0: If doc.Name <> "hexapodi.WM3" Then Exit Sub
  a = doc.Bodies("pb").GetConfig(pbx, pby, pbz, pbrx, pbry, pbrz)
  Open fl0 + "inp_inverse.txt" For Input As 1
  Input #1, xb, yb, zb, rxb, ryb, rzb: Close #1: zb = pbz0 + zb
  px = pbx * 1000: py = pby * 1000: pz = pbz * 1000
  rxa = pbrx * 180 / pi: rya = pbry * 180 / pi: rza = pbrz * 180 / pi
  Form1.Cls
  Form1.Print px, py, pz, rxa, rya, rza: Form1.Print xb, yb, zb, rxb, ryb, rzb
  xc = InputBox("Continue", , "y"): If xc <> "y" Then Exit Sub
  kp = 0: kpd = 50: aep = 0.0001: aer = 0.00001
  doc.RunTo 0: k = 0
  10 If k = ns Then MsgBox ("Convergence error"): GoTo 30
  pbx = doc.Meters("pbpos").Y1(k) * 1000
  pby = doc.Meters("pbpos").Y2(k) * 1000
  pbz = doc.Meters("pbpos").y3(k) * 1000
  rxa = doc.Meters("pbrot").Y1(k) * 180 / pi

```

```

rya = doc.Meters("pbrot").Y2(k) * 180 / pi
rza = doc.Meters("pbrot").y3(k) * 180 / pi
k = k + 1: kp = kp + kpd
epx = xb - pbx: epy = yb - pby: epz = zb - pbz
erx = rxb - rxa: ery = ryb - rya: erz = rzb - rza: kerror = 0
doc.Inputs("pbvx").value = kp * epx
doc.Inputs("pbvy").value = kp * epy
doc.Inputs("pbvz").value = kp * epz
doc.Inputs("pbwx").value = kp * erx
doc.Inputs("pbwy").value = kp * ery
doc.Inputs("pbwz").value = kp * erz
doc.RunTo k
If Abs(epx) > aep Then kerror = 1
If Abs(epy) > aep Then kerror = 1
If Abs(epz) > aep Then kerror = 1
If Abs(erx) > aer Then kerror = 1
If Abs(ery) > aer Then kerror = 1
If Abs(erz) > aer Then kerror = 1
If kerror = 0 Then GoTo 30
GoTo 10
30 doc.Inputs("pbvx").value = 0
doc.Inputs("pbvy").value = 0
doc.Inputs("pbvz").value = 0
doc.Inputs("pbwx").value = 0
doc.Inputs("pbwy").value = 0
doc.Inputs("pbwz").value = 0
doc.RunTo ns
End Sub

Sub rmeter1()           'Reads meters
  If konwm = 0 Then
    Set wm = GetObject("WM3D.Application"): Set doc = wm.ActiveDocument: Call const0
  End If
  n = ns: Form1.Cls
  a = doc.Bodies("pb").GetConfig(pbx, pby, pbz, pbrx, pbry, pbrz)
  Form1.Print pbx * 1000, pby * 1000, pbz * 1000
  Form1.Print pbrx * 180 / pi, pbry * 180 / pi, pbrz * 180 / pi
  Call cmm: Form1.Print
  If doc.NumFrames < ns Then

```

```

Form1.Print "ns=0"
doc.Inputs("pbvx").value = 0: doc.Inputs("pbvy").value = 0
doc.Inputs("pbvz").value = 0: doc.Inputs("pbwx").value = 0
doc.Inputs("pbwy").value = 0: doc.Inputs("pbwz").value = 0: n = 0: doc.RunTo 0
End If
ra(1) = doc.Meters("la1-l").Y1(n) - r0
ra(2) = doc.Meters("la2-l").Y1(n) - r0
ra(3) = doc.Meters("la3-l").Y1(n) - r0
ra(4) = doc.Meters("la4-l").Y1(n) - r0
ra(5) = doc.Meters("la5-l").Y1(n) - r0
ra(6) = doc.Meters("la6-l").Y1(n) - r0
If ns = 0 Then doc.Reset: doc.EraseHistory
Cx = Cos(pbrx): sx = Sin(pbrx): Cy = Cos(pbry): sy = Sin(pbry)
cz = Cos(pbrz): sz = Sin(pbrz): kerror = 0
For k = 1 To 6
lla = l1 + ra(k)
zz = (-Cx * sy * cz + sx * sz) * xsb(k) * 1000 + (Cx * sy * sz + sx * cz) * ysb(k) * 1000
zz = zz + Cx * Cy * zsb * 1000 + pbz * 1000: z = zz - zsa * 1000
x = Sqr(lla * lla - z ^ 2): fia = Atn(x / z) * 180 / pi
'---
x = (xsa(k) - pbx) * 1000: y = (ysa(k) - pby) * 1000: z = (zsa - pbz) * 1000
zz = sy * x - Cy * sx * y + Cy * Cx * z: z = -zz + zsb * 1000 '
x = Sqr(lla * lla - z ^ 2): fib = Atn(x / z) * 180 / pi
Form1.Print ra(k), fia, fib
If ra(k) < 0 Or ra(k) > 45 Then: kerror = 1
If fia > 35 Or fib > 35 Then: kerror = 1
Next k
If kerror = 1 Then MsgBox ("Error")
End Sub

```

```

Sub cmm()                                'Transformation
Cx = Cos(pbrx): Cy = Cos(pbry): cz = Cos(pbrz)
sx = Sin(pbrx): sy = Sin(pbry): sz = Sin(pbrz)
x = sy: y = -sx * Cy: z = Cx * Cy
cfi = x: GoSub 25: alpha = th: alphac = the
cfi = y: GoSub 25: beta = th: betac = the
cfi = z: GoSub 25: gama = th: gamac = the
Form1.Print
Form1.Print alphac, betac, gamac

```

```

Form1.Print alpha, beta, gama
Exit Sub
25 If cfi = 0 Then th = 90: Return
sfi = Sqr(1 - cfi * cfi): th = Atn(sfi / cfi) * 180 / pi
If th < 0 Then th = th + 180
30 thd = (th - Fix(th)) * 60: ths = Fix(((thd - Fix(thd)) * 60))
thd = Fix(thd): thc = Str(Fix(th)) + " " + Str(thd) + " " + Str(thc)
Return
End Sub

Sub forward1()           'Forward kinematics
    MsgBox ("Correct module"): Exit Sub
    Call m0: If doc.Name <> "hexapodf.WM3" Then Exit Sub
    Open fl0 + "inp_forward.txt" For Input As 1
    For k = 1 To 6: Input #1, la(k): Next k: Close #1
    Form1.Cls: Form1.Print la(1), la(2), la(3), la(4), la(5), la(6): MsgBox ("")
    doc.Inputs("la1").value = la(1) + r0
    doc.Inputs("la2").value = la(2) + r0
    doc.Inputs("la3").value = la(3) + r0
    doc.Inputs("la4").value = la(4) + r0
    doc.Inputs("la5").value = la(5) + r0
    doc.Inputs("la6").value = la(6) + r0
    doc.Run
End Sub

Sub m0()                 'Reset
    Call wm3d0: doc.Reset: doc.EraseHistory
    Call doc.Bodies("pb").SetInitialVelocity(0, 0, 0, 0, 0, 0)
End Sub

Sub cmm2dec()            'Arcmin to degree
    j = 3: If Mid(thc, 4, 1) = ":" Then j = 4
    th = Val(Mid(thc, 1, j - 1)) + Val(Mid(thc, j + 1, 2)) / 60 + Val(Mid(thc, j + 4, 2)) / 3600
End Sub

-----Move.bas-----

Public rp(6) As Double, disa(6) As Long
Public nmotor As Integer, dis0 As Double, dis As Double

```

```
Public v0p(6) As Long, t1p(6) As Double
```

```
Public v0puls As Long, t1 As Double
```

```
Public dismax As Long, dismin As Long
```

```
Sub move_motors()           'Control
```

```
    Form1.Print dis, t1, v0puls: GoTo 30
```

```
    If dis = 0 Then GoTo 30
```

```
    If nmotor = 1 Then naxis = 0: GoTo 10
```

```
    If nmotor = 2 Then naxis = 1: GoTo 10
```

```
    If nmotor = 3 Then naxis = 0: GoTo 20
```

```
    If nmotor = 4 Then naxis = 1: GoTo 20
```

```
    If nmotor = 5 Then naxis = 2: GoTo 20
```

```
    If nmotor = 6 Then naxis = 3: GoTo 20
```

```
    10 a = Form1.B_8132.StartTMove(naxis, dis, 0, v0puls, t1, t1): GoTo 30
```

```
    20 a = Form1.B_8164.StartTRMove(naxis, dis, 0, v0puls, t1, t1)
```

```
    30
```

```
End Sub
```

```
Sub move1()                 'Control
```

```
    v0max = 3000: t1max = 0.01
```

```
    If konwm = 0 Then
```

```
        Set wm = GetObject("WM3D.Application"): Set doc = wm.ActiveDocument: Call const0
```

```
    End If
```

```
    Call rmeter1: If kerror > 0 Then Exit Sub
```

```
    If doc.NumFrames < ns Then MsgBox ("ns=0"): Exit Sub
```

```
    rp(1) = doc.Meters("la1-l").Y1(0) - r0
```

```
    rp(2) = doc.Meters("la2-l").Y1(0) - r0
```

```
    rp(3) = doc.Meters("la3-l").Y1(0) - r0
```

```
    rp(4) = doc.Meters("la4-l").Y1(0) - r0
```

```
    rp(5) = doc.Meters("la5-l").Y1(0) - r0
```

```
    rp(6) = doc.Meters("la6-l").Y1(0) - r0
```

```
    For k = 1 To 6: disa(k) = CInt((ra(k) - rp(k)) / 0.00075): Next k
```

```
    dismax = Abs(disa(1))
```

```
    For k = 2 To 6
```

```
        If dismax < Abs(disa(k)) Then dismax = Abs(disa(k))
```

```
    Next k
```

```
    Form1.Print
```

```
    For k = 1 To 6
```

```
        dis = -disa(k): td = dismax / v0max + t1max: t2 = td - 2 * t1max: t1p(k) = t1max
```



```

If Abs(dis) < v0max * t1max Then t1p(k) = 0.5 * td: t2 = 0
v0p(k) = CInt(Abs(dis) / (t1p(k) + t2))
Form1.Print rp(k), ra(k), dis, t1p(k), v0p(k)
Next k
xc = InputBox("Continue", , "y"): If xc <> "y" Then Exit Sub
Call adlink1
End Sub

```

```

Sub adlink1()           'Control
  For nmotor = 1 To 6
    dis = -disa(nmotor): t1 = t1p(nmotor): v0puls = v0p(nmotor)
    Call move_motors
  Next nmotor
  doc.EraseHistory
End Sub

```

-----Adlink_init.bas-----

```

Sub adlink0()           'Initials cards
  Form1.B_8132.Axis0.OutputMode = OUT_DIR
  Form1.B_8132.Axis1.OutputMode = OUT_DIR
  Form1.B_8132.Axis0.InputMode = db4X_AB_PHASE
  Form1.B_8132.Axis1.InputMode = db4X_AB_PHASE
  '----
  Form1.B_8164.Axis0.OutputMode = OUT_RISING_DIR_HIGH
  Form1.B_8164.Axis1.OutputMode = OUT_RISING_DIR_HIGH
  Form1.B_8164.Axis2.OutputMode = OUT_RISING_DIR_HIGH
  Form1.B_8164.Axis3.OutputMode = OUT_RISING_DIR_HIGH
  Form1.B_8164.Axis0.InputMode = NORMAL_LOW_4X_AB_PHASE
  Form1.B_8164.Axis1.InputMode = NORMAL_LOW_4X_AB_PHASE
  Form1.B_8164.Axis2.InputMode = NORMAL_LOW_4X_AB_PHASE
  Form1.B_8164.Axis3.InputMode = NORMAL_LOW_4X_AB_PHASE
End Sub

```

APPENDIX E
VISUALBASIC PROGRAM FOR CONTROLLING OF SERVO MOTOR
SYSTEMS

```
Dim fl0 As String
```

```
Private Sub Command1_Click()  
    Call path  
End Sub
```

```
Private Sub Command10_Click()  
    Call mot2  
End Sub
```

```
Private Sub Command11_Click()  
    Call mot3  
End Sub
```

```
Private Sub Command12_Click()  
    Call mot4  
End Sub
```

```
Private Sub Command13_Click()  
    Call mot5  
End Sub
```

```
Private Sub Command14_Click()  
    Call mot7  
End Sub
```

```
Private Sub Command2_Click()  
    Call savep  
End Sub
```

```
Private Sub Command3_Click()  
    Call move1  
End Sub
```

```
Private Sub Command4_Click()
```

```
    Call readp
```

```
End Sub
```

```
Private Sub Command5_Click()
```

```
    Call move3li
```

```
End Sub
```

```
Private Sub Command6_Click()
```

```
    Call pnt0
```

```
End Sub
```

```
Private Sub Command7_Click()
```

```
    Call stop1
```

```
End Sub
```

```
Private Sub Command8_Click()
```

```
    Call mot6
```

```
End Sub
```

```
Private Sub Command9_Click()
```

```
    Call mot1
```

```
End Sub
```

```
Private Sub Form_Load()
```

```
    fl0 = "d:\r06\": ' fl0 = app.path + "\"
```

```
    WindowState = 2
```

```
    Form1.Caption = "PC-BASED MOTION CONTROL"
```

```
    Command1.Caption = "Get Vel.": Command2.Caption = "End"
```

```
    Command3.Caption = "Move1": Command4.Caption = "Read Pos."
```

```
    Command5.Caption = "Move3_li": Command6.Caption = "Test"
```

```
    Command7.Caption = "Stop": Command8.Caption = "Closed loop with abs."
```

```
    Command9.Caption = "Mot with clk": Command10.Caption = "Mot with sleep"
```

```
    Command11.Caption = "Mot. with pos control"
```

```
    Command12.Caption = "Mot. with closed loop pos control"
```

```
    Command13.Caption = "Mot. with closed loop pos and vel control"
```

```
    Command14.Caption = "Closed loop with rel."
```

```
    Text1.Text = "": Text5.Text = "": Text6.Text = ""
```

```
    Picture1.AutoRedraw = True: Picture2.AutoRedraw = True: Picture3.AutoRedraw = True
```

```

Form1.AutoRedraw = True
a = B_8164_set_feedback_src(0, 0): a = B_8164_set_feedback_src(1, 0): a =
B_8164_set_feedback_src(2, 0)
B_8164.Axis0.OutputMode = OUT_RISING_DIR_LOW
B_8164.Axis1.OutputMode = OUT_RISING_DIR_LOW
B_8164.Axis2.OutputMode = OUT_RISING_DIR_LOW
End Sub

```

```

Private Sub Timer1_Timer()
Dim fb0 As Double: Dim fb1 As Double: Dim fb2 As Double
Dim vb0 As Double: Dim vb1 As Double: Dim vb2 As Double
a = B_8164_get_position(0, fb0): a = B_8164_get_position(1, fb1): a =
B_8164_get_position(2, fb2)
fb0 = fb0 * 360 / nenc / ngear: fb1 = fb1 * 360 / nenc / ngear: fb2 = fb2 * 360 / nenc / ngear
Text2.Text = fb0: Text3.Text = fb1: Text4.Text = fb2
a = Form1.B_8164.GetCurrentSpeed(0, vb0): a = Form1.B_8164.GetCurrentSpeed(1, vb1): a =
Form1.B_8164.GetCurrentSpeed(2, vb2)
vb0 = vb0 * 360 / nenc / ngear: vb1 = vb1 * 360 / nenc / ngear: vb2 = vb2 * 360 / nenc / ngear
Text7.Text = vb0: Text8.Text = vb1: Text9.Text = vb2
End Sub

```

----Module----

```

Public Const nmax1 = 256, pi = 3.141593, nenc = 2048, ngear = 74, Tacc = 0.001
Public fl0 As String: Public a, ns As Integer
Public yr0(nmax1), yr1(nmax1), yr2(nmax1) As Single
Public yrm0(nmax1), yrm1(nmax1), yrm2(nmax1) As Single
Public dt, tr(nmax1), thr0(nmax1), ang0(nmax1), thr1(nmax1), ang1(nmax1), thr2(nmax1),
ang2(nmax1) As Single
Public npuls0(nmax1), npuls1(nmax1), npuls2(nmax1)
Public pos0(nmax1) As Double: Public pos1(nmax1) As Double: Public pos2(nmax1) As Double
Public v0(nmax1) As Double: Public v1(nmax1) As Double: Public v2(nmax1) As Double
Public perr0(nmax1), perr1(nmax1), perr2(nmax1), verr0(nmax1), verr1(nmax1), verr2(nmax1) As
Double
Public kperr0(nmax1), kperr1(nmax1), kperr2(nmax1), kverr0(nmax1), kverr1(nmax1),
kverr2(nmax1) As Double
Public kang0(nmax1), kang1(nmax1), kang2(nmax1) As Double
Public ftim0 As Double: Public tim0(nmax1) As Double: Public tim1(nmax1) As Double: Public
tim2(nmax1) As Double
Declare Sub Sleep Lib "kernel32" (ByVal dwMilliseconds As Long)

```

Sub mot7('Algorithm 7

Form1.Cls: Call path

coe = InputBox("pos. gain"): If coe = "" Then Exit Sub

Call initialm

perr0(1) = ang0(1) - pos0(0): kperr0(1) = coe * perr0(1)

perr1(1) = ang1(1) - pos1(0): kperr1(1) = coe * perr1(1)

perr2(1) = ang2(1) - pos2(0): kperr2(1) = coe * perr2(1)

kang0(1) = ang0(2) - ang0(1) + kperr0(1)

kang1(1) = ang1(2) - ang1(1) + kperr1(1)

kang2(1) = ang2(2) - ang2(1) + kperr2(1)

a = Form1.B_8164.StartTRMove(0, kang0(1), npuls0(1), npuls0(2), dt, 0)

a = Form1.B_8164.StartTRMove(1, kang1(1), npuls1(1), npuls1(2), dt, 0)

a = Form1.B_8164.StartTRMove(2, kang2(1), npuls2(1), npuls2(2), dt, 0)

10 a = Form1.B_8164.GetGeneralCounter(0, tim0(1)): a = Form1.B_8164.GetGeneralCounter(1, tim1(1)): a = Form1.B_8164.GetGeneralCounter(2, tim2(1))

ftim0 = tim0(1) - tim0(0): 'If ftim0 > (tim1(1) - tim1(0)) Then ftim0 = (tim1(1) - tim1(0)): If ftim0 > (tim2(1) - tim2(0)) Then ftim0 = (tim2(1) - tim2(0))

If ftim0 < dt * 10000000 Then GoTo 10

a = Form1.B_8164.GetCurrentSpeed(0, v0(1)): a = Form1.B_8164.GetCurrentSpeed(1, v1(1)): a = Form1.B_8164.GetCurrentSpeed(2, v2(1))

a = Form1.B_8164.GetPosition(0, pos0(1)): a = Form1.B_8164.GetPosition(1, pos1(1)): a = Form1.B_8164.GetPosition(2, pos2(1))

For k = 2 To ns - 1

perr0(k) = ang0(k) - pos0(k - 1): kperr0(k) = coe * perr0(k)

perr1(k) = ang1(k) - pos1(k - 1): kperr1(k) = coe * perr1(k)

perr2(k) = ang2(k) - pos2(k - 1): kperr2(k) = coe * perr2(k)

kang0(k) = ang0(k + 1) - ang0(k) + kperr0(k)

kang1(k) = ang1(k + 1) - ang1(k) + kperr1(k)

kang2(k) = ang2(k + 1) - ang2(k) + kperr2(k)

a = Form1.B_8164.StartTRMove(0, kang0(k), npuls0(k), npuls0(k + 1), dt, 0)

a = Form1.B_8164.StartTRMove(1, kang1(k), npuls1(k), npuls1(k + 1), dt, 0)

a = Form1.B_8164.StartTRMove(2, kang2(k), npuls2(k), npuls2(k + 1), dt, 0)

20 a = Form1.B_8164.GetGeneralCounter(0, tim0(k)): a = Form1.B_8164.GetGeneralCounter(1, tim1(k)): a = Form1.B_8164.GetGeneralCounter(2, tim2(k))

ftim0 = tim0(k) - tim0(k - 1): 'If ftim0 > (tim1(k) - tim1(k - 1)) Then ftim0 = (tim1(k) - tim1(k - 1)): If ftim0 > (tim2(k) - tim2(k - 1)) Then ftim0 = (tim2(k) - tim2(k - 1))

If ftim0 < dt * 10000000 Then GoTo 20

a = Form1.B_8164.GetCurrentSpeed(0, v0(k)): a = Form1.B_8164.GetCurrentSpeed(1, v1(k)): a = Form1.B_8164.GetCurrentSpeed(2, v2(k))

```

a = Form1.B_8164.GetPosition(0, pos0(k)): a = Form1.B_8164.GetPosition(1, pos1(k)): a =
Form1.B_8164.GetPosition(2, pos2(k))
Next k
Call closem: Call plot2: Call write1: MsgBox ("")
End Sub

```

```

Sub mot6()                                'Algorithm 6
Form1.Cls: Call path
coe = InputBox("pos. gain"): If coe = "" Then Exit Sub
Call initialm
perr0(1) = ang0(1) - pos0(0): kperr0(1) = coe * perr0(1)
perr1(1) = ang1(1) - pos1(0): kperr1(1) = coe * perr1(1)
perr2(1) = ang2(1) - pos2(0): kperr2(1) = coe * perr2(1)
kang0(1) = ang0(2) + kperr0(1)
kang1(1) = ang1(2) + kperr1(1)
kang2(1) = ang2(2) + kperr2(1)
a = Form1.B_8164.StartTAMove(0, kang0(1), npuls0(1), npuls0(2), dt, 0)
a = Form1.B_8164.StartTAMove(1, kang1(1), npuls1(1), npuls1(2), dt, 0)
a = Form1.B_8164.StartTAMove(2, kang2(1), npuls2(1), npuls2(2), dt, 0)
10 a = Form1.B_8164.GetGeneralCounter(0, tim0(1)): a = Form1.B_8164.GetGeneralCounter(1,
tim1(1)): a = Form1.B_8164.GetGeneralCounter(2, tim2(1))
ftim0 = tim0(1) - tim0(0): 'If ftim0 > (tim1(1) - tim1(0)) Then ftim0 = (tim1(1) - tim1(0)): If
ftim0 > (tim2(1) - tim2(0)) Then ftim0 = (tim2(1) - tim2(0))
If ftim0 < dt * 10000000 Then GoTo 10
a = Form1.B_8164.GetCurrentSpeed(0, v0(1)): a = Form1.B_8164.GetCurrentSpeed(1, v1(1)):
a = Form1.B_8164.GetCurrentSpeed(2, v2(1))
a = Form1.B_8164.GetPosition(0, pos0(1)): a = Form1.B_8164.GetPosition(1, pos1(1)): a =
Form1.B_8164.GetPosition(2, pos2(1))
For k = 2 To ns - 1
perr0(k) = ang0(k) - pos0(k - 1): kperr0(k) = coe * perr0(k)
perr1(k) = ang1(k) - pos1(k - 1): kperr1(k) = coe * perr1(k)
perr2(k) = ang2(k) - pos2(k - 1): kperr2(k) = coe * perr2(k)
kang0(k) = ang0(k + 1) + kperr0(k)
kang1(k) = ang1(k + 1) + kperr1(k)
kang2(k) = ang2(k + 1) + kperr2(k)
a = Form1.B_8164.StartTAMove(0, kang0(k), npuls0(k), npuls0(k + 1), dt, 0)
a = Form1.B_8164.StartTAMove(1, kang1(k), npuls1(k), npuls1(k + 1), dt, 0)
a = Form1.B_8164.StartTAMove(2, kang2(k), npuls2(k), npuls2(k + 1), dt, 0)

```

```

20 a = Form1.B_8164.GetGeneralCounter(0, tim0(k)): 'a = Form1.B_8164.GetGeneralCounter(1,
tim1(k)): a = Form1.B_8164.GetGeneralCounter(2, tim2(k))
    ftim0 = tim0(k) - tim0(k - 1): 'If ftim0 > (tim1(k) - tim1(k - 1)) Then ftim0 = (tim1(k) - tim1(k
- 1)): If ftim0 > (tim2(k) - tim2(k - 1)) Then ftim0 = (tim2(k) - tim2(k - 1))
    If ftim0 < dt * 10000000 Then GoTo 20
    a = Form1.B_8164.GetCurrentSpeed(0, v0(k)): a = Form1.B_8164.GetCurrentSpeed(1, v1(k)):
a = Form1.B_8164.GetCurrentSpeed(2, v2(k))
    a = Form1.B_8164.GetPosition(0, pos0(k)): a = Form1.B_8164.GetPosition(1, pos1(k)): a =
Form1.B_8164.GetPosition(2, pos2(k))
    Next k
    Call closen: Call plot2: Call write1: MsgBox ("")
End Sub

```

```

Sub mot5()                                'Algorithm 5
    Form1.Cls: Call path
    coep = InputBox("pos. gain"): If coep = "" Then Exit Sub
    coev = InputBox("vel. gain"): If coev = "" Then Exit Sub
    Call initialm
    npuls0(1) = 0: npuls1(1) = 0: npuls2(1) = 0
    perr0(1) = ang0(2) - pos0(0): kperr0(1) = coep * perr0(1)
    perr1(1) = ang1(2) - pos1(0): kperr1(1) = coep * perr1(1)
    perr2(1) = ang2(2) - pos2(0): kperr2(1) = coep * perr2(1)
    verr0(1) = npuls0(2) - v0(0): If npuls0(2) < 0 Then verr0(1) = npuls0(2) + v0(0): kverr0(1) =
coev * verr0(1)
    verr1(1) = npuls1(2) - v1(0): If npuls1(2) < 0 Then verr1(1) = npuls1(2) + v1(0): kverr1(1) =
coev * verr1(1)
    verr2(1) = npuls2(2) - v2(0): If npuls2(2) < 0 Then verr2(1) = npuls2(2) + v2(0): kverr2(1) =
coev * verr2(1)
    a = Form1.B_8164.StartTAMove(0, kperr0(1), npuls0(1), kverr0(1), dt, 0)
    a = Form1.B_8164.StartTAMove(1, kperr1(1), npuls1(1), kverr1(1), dt, 0)
    a = Form1.B_8164.StartTAMove(2, kperr2(1), npuls2(1), kverr2(1), dt, 0)
10 a = Form1.B_8164.GetGeneralCounter(0, tim0(1)): 'a = Form1.B_8164.GetGeneralCounter(1,
tim1(1)): a = Form1.B_8164.GetGeneralCounter(2, tim2(1))
    ftim0 = tim0(1) - tim0(0): 'If ftim0 > (tim1(1) - tim1(0)) Then ftim0 = (tim1(1) - tim1(0)): If
ftim0 > (tim2(1) - tim2(0)) Then ftim0 = (tim2(1) - tim2(0))
    If ftim0 < dt * 10000000 Then GoTo 10
    a = Form1.B_8164.GetCurrentSpeed(0, v0(1)): a = Form1.B_8164.GetCurrentSpeed(1, v1(1)):
a = Form1.B_8164.GetCurrentSpeed(2, v2(1))

```

```

a = Form1.B_8164.GetPosition(0, pos0(1)): a = Form1.B_8164.GetPosition(1, pos1(1)): a =
Form1.B_8164.GetPosition(2, pos2(1))
For k = 2 To ns - 1
perr0(k) = ang0(k + 1) - pos0(k - 1): kperr0(k) = coep * perr0(k)
perr1(k) = ang1(k + 1) - pos1(k - 1): kperr1(k) = coep * perr1(k)
perr2(k) = ang2(k + 1) - pos2(k - 1): kperr2(k) = coep * perr2(k)
verr0(k) = npuls0(k + 1) - v0(k - 1): If npuls0(k + 1) < 0 Then verr0(k) = npuls0(k + 1) + v0(k -
1): kverr0(k) = coev * verr0(k)
verr1(k) = npuls1(k + 1) - v1(k - 1): If npuls1(k + 1) < 0 Then verr1(k) = npuls1(k + 1) + v1(k -
1): kverr1(k) = coev * verr1(k)
verr2(k) = npuls2(k + 1) - v2(k - 1): If npuls2(k + 1) < 0 Then verr2(k) = npuls2(k + 1) + v2(k -
1): kverr2(k) = coev * verr2(k)
a = Form1.B_8164.StartTAMove(0, kperr0(k), kverr0(k - 1), kverr0(k), dt, 0)
a = Form1.B_8164.StartTAMove(1, kperr1(k), kverr1(k - 1), kverr1(k), dt, 0)
a = Form1.B_8164.StartTAMove(2, kperr2(k), kverr2(k - 1), kverr2(k), dt, 0)
20 a = Form1.B_8164.GetGeneralCounter(0, tim0(k)): a = Form1.B_8164.GetGeneralCounter(1,
tim1(k)): a = Form1.B_8164.GetGeneralCounter(2, tim2(k))
ftim0 = tim0(k) - tim0(k - 1): ' If ftim0 > (tim1(k) - tim1(k - 1)) Then ftim0 = (tim1(k) - tim1(k
- 1)): If ftim0 > (tim2(k) - tim2(k - 1)) Then ftim0 = (tim2(k) - tim2(k - 1))
If ftim0 < dt * 10000000 Then GoTo 20
a = Form1.B_8164.GetCurrentSpeed(0, v0(k)): a = Form1.B_8164.GetCurrentSpeed(1, v1(k)):
a = Form1.B_8164.GetCurrentSpeed(2, v2(k))
a = Form1.B_8164.GetPosition(0, pos0(k)): a = Form1.B_8164.GetPosition(1, pos1(k)): a =
Form1.B_8164.GetPosition(2, pos2(k))
Next k
Call closen: Call plot2: Call write1: MsgBox ("")
End Sub

```

Sub mot4() 'Algorithm 4

```

Form1.Cls: Call path
coe = InputBox("pos. gain"): If coe = "" Then Exit Sub
Call initialm
perr0(1) = ang0(2) - pos0(0): kperr0(1) = coe * perr0(1)
perr1(1) = ang1(2) - pos1(0): kperr1(1) = coe * perr1(1)
perr2(1) = ang2(2) - pos2(0): kperr2(1) = coe * perr2(1)
a = Form1.B_8164.StartTAMove(0, kperr0(1), npuls0(1), npuls0(2), dt, 0)
a = Form1.B_8164.StartTAMove(1, kperr1(1), npuls1(1), npuls1(2), dt, 0)
a = Form1.B_8164.StartTAMove(2, kperr2(1), npuls2(1), npuls2(2), dt, 0)

```



```

10 a = Form1.B_8164.GetGeneralCounter(0, tim0(1)): 'a = Form1.B_8164.GetGeneralCounter(1,
tim1(1)): a = Form1.B_8164.GetGeneralCounter(2, tim2(1))
    ftim0 = tim0(1) - tim0(0): 'If ftim0 > (tim1(1) - tim1(0)) Then ftim0 = (tim1(1) - tim1(0)): If
ftim0 > (tim2(1) - tim2(0)) Then ftim0 = (tim2(1) - tim2(0))
    If ftim0 < dt * 10000000 Then GoTo 10
    a = Form1.B_8164.GetCurrentSpeed(0, v0(1)): a = Form1.B_8164.GetCurrentSpeed(1, v1(1)):
a = Form1.B_8164.GetCurrentSpeed(2, v2(1))
    a = Form1.B_8164.GetPosition(0, pos0(1)): a = Form1.B_8164.GetPosition(1, pos1(1)): a =
Form1.B_8164.GetPosition(2, pos2(1))
    For k = 2 To ns - 1
    perr0(k) = ang0(k + 1) - pos0(k - 1): kperr0(k) = coe * perr0(k)
    perr1(k) = ang1(k + 1) - pos1(k - 1): kperr1(k) = coe * perr1(k)
    perr2(k) = ang2(k + 1) - pos2(k - 1): kperr2(k) = coe * perr2(k)
    a = Form1.B_8164.StartTAMove(0, kperr0(k), npuls0(k), npuls0(k + 1), dt, 0)
    a = Form1.B_8164.StartTAMove(1, kperr1(k), npuls1(k), npuls1(k + 1), dt, 0)
    a = Form1.B_8164.StartTAMove(2, kperr2(k), npuls2(k), npuls2(k + 1), dt, 0)
20 a = Form1.B_8164.GetGeneralCounter(0, tim0(k)): 'a = Form1.B_8164.GetGeneralCounter(1,
tim1(k)): a = Form1.B_8164.GetGeneralCounter(2, tim2(k))
    ftim0 = tim0(k) - tim0(k - 1): 'If ftim0 > (tim1(k) - tim1(k - 1)) Then ftim0 = (tim1(k) - tim1(k
- 1)): If ftim0 > (tim2(k) - tim2(k - 1)) Then ftim0 = (tim2(k) - tim2(k - 1))
    If ftim0 < dt * 10000000 Then GoTo 20
    a = Form1.B_8164.GetCurrentSpeed(0, v0(k)): a = Form1.B_8164.GetCurrentSpeed(1, v1(k)):
a = Form1.B_8164.GetCurrentSpeed(2, v2(k))
    a = Form1.B_8164.GetPosition(0, pos0(k)): a = Form1.B_8164.GetPosition(1, pos1(k)): a =
Form1.B_8164.GetPosition(2, pos2(k))
    Next k
    Call closem: Call plot2: Call write1: MsgBox ("")
End Sub

```

```

Sub mot3() 'Algorithm 3

```

```

    Form1.Cls: Call path: Call initialm

```

```

    a = Form1.B_8164.StartTAMove(0, ang0(2), npuls0(1), npuls0(2), dt, 0)

```

```

    a = Form1.B_8164.StartTAMove(1, ang1(2), npuls1(1), npuls1(2), dt, 0)

```

```

    a = Form1.B_8164.StartTAMove(2, ang2(2), npuls2(1), npuls2(2), dt, 0)

```

```

10 a = Form1.B_8164.GetGeneralCounter(0, tim0(1)): 'a = Form1.B_8164.GetGeneralCounter(1,
tim1(1)): a = Form1.B_8164.GetGeneralCounter(2, tim2(1))

```

```

    ftim0 = tim0(1) - tim0(0): 'If ftim0 > (tim1(1) - tim1(0)) Then ftim0 = (tim1(1) - tim1(0)): If
ftim0 > (tim2(1) - tim2(0)) Then ftim0 = (tim2(1) - tim2(0))

```

```

    If ftim0 < dt * 10000000 Then GoTo 10

```

```

a = Form1.B_8164.GetCurrentSpeed(0, v0(1)): a = Form1.B_8164.GetCurrentSpeed(1, v1(1)):
a = Form1.B_8164.GetCurrentSpeed(2, v2(1))
a = Form1.B_8164.GetPosition(0, pos0(1)): a = Form1.B_8164.GetPosition(1, pos1(1)): a =
Form1.B_8164.GetPosition(2, pos2(1))
For k = 2 To ns - 1
a = Form1.B_8164.StartTAMove(0, ang0(k + 1), npuls0(k), npuls0(k + 1), dt, 0)
a = Form1.B_8164.StartTAMove(1, ang1(k + 1), npuls1(k), npuls1(k + 1), dt, 0)
a = Form1.B_8164.StartTAMove(2, ang2(k + 1), npuls2(k), npuls2(k + 1), dt, 0)
20 a = Form1.B_8164.GetGeneralCounter(0, tim0(k)): a = Form1.B_8164.GetGeneralCounter(1,
tim1(k)): a = Form1.B_8164.GetGeneralCounter(2, tim2(k))
ftim0 = tim0(k) - tim0(k - 1): ' If ftim0 > (tim1(k) - tim1(k - 1)) Then ftim0 = (tim1(k) - tim1(k
- 1)): If ftim0 > (tim2(k) - tim2(k - 1)) Then ftim0 = (tim2(k) - tim2(k - 1))
If ftim0 < dt * 10000000 Then GoTo 20
a = Form1.B_8164.GetCurrentSpeed(0, v0(k)): a = Form1.B_8164.GetCurrentSpeed(1, v1(k)):
a = Form1.B_8164.GetCurrentSpeed(2, v2(k))
a = Form1.B_8164.GetPosition(0, pos0(k)): a = Form1.B_8164.GetPosition(1, pos1(k)): a =
Form1.B_8164.GetPosition(2, pos2(k))
Next k
Call closen: Call plot2: Call write1: MsgBox ("")
End Sub

```

```

Sub mot2() 'Algorithm 2
Form1.Cls: Call path: Call initialm
a = Form1.B_8164.TVMove(0, npuls0(1), npuls0(2), dt)
a = Form1.B_8164.TVMove(1, npuls1(1), npuls1(2), dt)
a = Form1.B_8164.TVMove(2, npuls2(1), npuls2(2), dt)
Sleep (dt * 1000)
a = Form1.B_8164.GetCurrentSpeed(0, v0(1)): a = Form1.B_8164.GetCurrentSpeed(1, v1(1)):
a = Form1.B_8164.GetCurrentSpeed(2, v2(1))
a = Form1.B_8164.GetPosition(0, pos0(1)): a = Form1.B_8164.GetPosition(1, pos1(1)): a =
Form1.B_8164.GetPosition(2, pos2(1))
For k = 2 To ns - 1
a = Form1.B_8164.VChange(0, npuls0(k + 1), dt)
a = Form1.B_8164.VChange(1, npuls1(k + 1), dt)
a = Form1.B_8164.VChange(2, npuls2(k + 1), dt)
Sleep (dt * 1000)
a = Form1.B_8164.GetCurrentSpeed(0, v0(k)): a = Form1.B_8164.GetCurrentSpeed(1, v1(k)):
a = Form1.B_8164.GetCurrentSpeed(2, v2(k))

```

```

a = Form1.B_8164.GetPosition(0, pos0(k)): a = Form1.B_8164.GetPosition(1, pos1(k)): a =
Form1.B_8164.GetPosition(2, pos2(k))
If npuls0(k) > 0 And npuls0(k + 1) < 0 Then Form1.B_8164.Axis0.OutputMode =
OUT_RISING_DIR_HIGH
If npuls0(k) < 0 And npuls0(k + 1) > 0 Then Form1.B_8164.Axis0.OutputMode =
OUT_RISING_DIR_HIGH
If npuls1(k) > 0 And npuls1(k + 1) < 0 Then Form1.B_8164.Axis1.OutputMode =
OUT_RISING_DIR_HIGH
If npuls1(k) < 0 And npuls1(k + 1) > 0 Then Form1.B_8164.Axis1.OutputMode =
OUT_RISING_DIR_HIGH
If npuls2(k) > 0 And npuls2(k + 1) < 0 Then Form1.B_8164.Axis2.OutputMode =
OUT_RISING_DIR_HIGH
If npuls2(k) < 0 And npuls2(k + 1) > 0 Then Form1.B_8164.Axis2.OutputMode =
OUT_RISING_DIR_HIGH
Next k
Call closen: Call plot2: Call write1: MsgBox ("")
End Sub

```

```

Sub mot1()           'Algorithm 1
Form1.Cls: Call path: Call initialm
a = Form1.B_8164.TVMove(0, npuls0(1), npuls0(2), dt)
a = Form1.B_8164.TVMove(1, npuls1(1), npuls1(2), dt)
a = Form1.B_8164.TVMove(2, npuls2(1), npuls2(2), dt)
10 a = Form1.B_8164.GetGeneralCounter(0, tim0(1)): ' a = Form1.B_8164.GetGeneralCounter(1,
tim1(1)): a = Form1.B_8164.GetGeneralCounter(2, tim2(1))
ftim0 = tim0(1) - tim0(0): 'If ftim0 > (tim1(1) - tim1(0)) Then ftim0 = (tim1(1) - tim1(0)): If
ftim0 > (tim2(1) - tim2(0)) Then ftim0 = (tim2(1) - tim2(0))
If ftim0 < dt * 10000000 Then GoTo 10
a = Form1.B_8164.GetCurrentSpeed(0, v0(1)): a = Form1.B_8164.GetCurrentSpeed(1, v1(1)):
a = Form1.B_8164.GetCurrentSpeed(2, v2(1))
a = Form1.B_8164.GetPosition(0, pos0(1)): a = Form1.B_8164.GetPosition(1, pos1(1)): a =
Form1.B_8164.GetPosition(2, pos2(1))
For k = 2 To ns - 1
a = Form1.B_8164.VChange(0, npuls0(k + 1), dt): a = Form1.B_8164.VChange(1, npuls1(k +
1), dt): a = Form1.B_8164.VChange(2, npuls2(k + 1), dt)
20 a = Form1.B_8164.GetGeneralCounter(0, tim0(k)): ' a = Form1.B_8164.GetGeneralCounter(1,
tim1(k)): a = Form1.B_8164.GetGeneralCounter(2, tim2(k))
ftim0 = tim0(k) - tim0(k - 1): ' If ftim0 > (tim1(k) - tim1(k - 1)) Then ftim0 = (tim1(k) - tim1(k
- 1)): If ftim0 > (tim2(k) - tim2(k - 1)) Then ftim0 = (tim2(k) - tim2(k - 1))

```

```

If ftim0 < dt * 10000000 Then GoTo 20
a = Form1.B_8164.GetCurrentSpeed(0, v0(k)): a = Form1.B_8164.GetCurrentSpeed(1, v1(k)):
a = Form1.B_8164.GetCurrentSpeed(2, v2(k))
a = Form1.B_8164.GetPosition(0, pos0(k)): a = Form1.B_8164.GetPosition(1, pos1(k)): a =
Form1.B_8164.GetPosition(2, pos2(k))
If npuls0(k) > 0 And npuls0(k + 1) < 0 Then Form1.B_8164.Axis0.OutputMode =
OUT_RISING_DIR_HIGH
If npuls0(k) < 0 And npuls0(k + 1) > 0 Then Form1.B_8164.Axis0.OutputMode =
OUT_RISING_DIR_HIGH
If npuls1(k) > 0 And npuls1(k + 1) < 0 Then Form1.B_8164.Axis1.OutputMode =
OUT_RISING_DIR_HIGH
If npuls1(k) < 0 And npuls1(k + 1) > 0 Then Form1.B_8164.Axis1.OutputMode =
OUT_RISING_DIR_HIGH
If npuls2(k) > 0 And npuls2(k + 1) < 0 Then Form1.B_8164.Axis2.OutputMode =
OUT_RISING_DIR_HIGH
If npuls2(k) < 0 And npuls2(k + 1) > 0 Then Form1.B_8164.Axis2.OutputMode =
OUT_RISING_DIR_HIGH
Next k
Call closen: Call plot2: Call write1: MsgBox ("")
End Sub

```

```

Sub initialm()           'Initials algorithms
a = Form1.B_8164.SetContinuousMove(0, 1): a = Form1.B_8164.SetContinuousMove(1, 1): a
= Form1.B_8164.SetContinuousMove(2, 1)
a = Form1.B_8164.FixSpeedRange(0, 100000): a = Form1.B_8164.FixSpeedRange(1,
100000): a = Form1.B_8164.FixSpeedRange(2, 100000)
a = Form1.B_8164.SetGeneralCounter(0, 3, 0): a = Form1.B_8164.SetGeneralCounter(1, 3, 0):
a = Form1.B_8164.SetGeneralCounter(2, 3, 0)
a = Form1.B_8164.SetPosition(0, ang0(1)): a = Form1.B_8164.SetPosition(1, ang1(1)): a =
Form1.B_8164.SetPosition(2, ang2(1))
a = Form1.B_8164.GetCurrentSpeed(0, v0(0)): a = Form1.B_8164.GetCurrentSpeed(1, v1(0)):
a = Form1.B_8164.GetCurrentSpeed(2, v2(0))
a = Form1.B_8164.GetPosition(0, pos0(0)): a = Form1.B_8164.GetPosition(1, pos1(0)): a =
Form1.B_8164.GetPosition(2, pos2(0))
a = Form1.B_8164.GetGeneralCounter(0, tim0(0)): a = Form1.B_8164.GetGeneralCounter(1,
tim1(0)): a = Form1.B_8164.GetGeneralCounter(2, tim2(0))
End Sub

```

```

Sub closेम()                'Closes algorithms
    a = Form1.B_8164.SDStop(0, Tacc): a = Form1.B_8164.SDStop(1, Tacc): a =
Form1.B_8164.SDStop(2, Tacc)
    a = Form1.B_8164.GetGeneralCounter(0, tim0(ns)): a = Form1.B_8164.GetGeneralCounter(1,
tim1(ns)): a = Form1.B_8164.GetGeneralCounter(2, tim2(ns))
    ftim0 = tim0(ns) - tim0(0): If ftim0 < (tim1(ns) - tim1(0)) Then ftim0 = (tim1(ns) - tim1(0)): If
ftim0 < (tim2(ns) - tim2(0)) Then ftim0 = (tim2(ns) - tim2(0))
    a = Form1.B_8164.GetCurrentSpeed(0, v0(ns)): a = Form1.B_8164.GetCurrentSpeed(1,
v1(ns)): a = Form1.B_8164.GetCurrentSpeed(2, v2(ns))
    a = Form1.B_8164.UnFixSpeedRange(0): a = Form1.B_8164.UnFixSpeedRange(1): a =
Form1.B_8164.UnFixSpeedRange(2)
    Form1.Print ftim0
    a = Form1.B_8164.GetPosition(0, pos0(ns)): a = Form1.B_8164.GetPosition(1, pos1(ns)): a =
Form1.B_8164.GetPosition(2, pos2(ns))
    For k = 1 To ns: yr0(k) = ang0(k): yr1(k) = ang1(k): yr2(k) = ang2(k): Next k
    For k = 1 To ns: yrm0(k) = pos0(k - 1): yrm1(k) = pos1(k - 1): yrm2(k) = pos2(k - 1): Next k
End Sub

```

```

Sub write1()                'Writes data into a file
    fl0 = "d:\r06\": flth = fl0 + "r06wr.txt"
    Open flth For Output As 1
    t = 0
    For k = 1 To ns
    Print #1, Str(t), Str(yr0(k)), Str(yrm0(k)), Str(yr1(k)), Str(yrm1(k)), Str(yr2(k)), Str(yrm2(k))
    t = t + dt
    Next k
    Close #1
End Sub

```

```

Sub path()                  'Creates reference curve data
    fl0 = "d:\r06\"
    flth = fl0 + "r06p.txt": Open flth For Input As 1
    For k = 1 To 4: Line Input #1, xc: Next k: k = 0
    10 If EOF(1) = -1 Then GoTo 20
    k = k + 1: Input #1, t, xc, xc, th0, xc, xc, xc, xc, th1, xc, xc, xc, xc, th2, xc, xc, xc, xc, r0, xc,
xc, xc, r1, xc, xc, xc, r2
    tr(k) = t: thr0(k) = th0: ang0(k) = r0: thr1(k) = th1: ang1(k) = r1: thr2(k) = th2: ang2(k) = r2
    GoTo 10
    20 Close #1: ns = k

```

```

dt = tr(2) - tr(1)
For k = 1 To ns: npuls0(k) = thr0(k) * ngear * nenc / 360: npuls1(k) = thr1(k) * ngear * nenc /
360: npuls2(k) = thr2(k) * ngear * nenc / 360: Next k 'npuls velocity
For k = 1 To ns: ang0(k) = ang0(k) * ngear * nenc / 360: ang1(k) = ang1(k) * ngear * nenc /
360: ang2(k) = ang2(k) * ngear * nenc / 360: Next k 'ang position
Form1.B_8164.Axis0.OutputMode = OUT_RISING_DIR_LOW
Form1.B_8164.Axis1.OutputMode = OUT_RISING_DIR_LOW
Form1.B_8164.Axis2.OutputMode = OUT_RISING_DIR_LOW
End Sub

Sub stop1() 'Stops all axes
a = Form1.B_8164.SDStop(0, 0.001)
a = Form1.B_8164.SDStop(1, 0.001)
a = Form1.B_8164.SDStop(2, 0.001)
End Sub

Sub move3li() 'Three axes linear interpolation
Dim ax(2) As Integer
ngear = 74: th01 = 0: th02 = -30: th11 = 0: th12 = -40: th21 = 0: th22 = -50: dt = 1
th0 = th02 - th01: th1 = th12 - th11: th2 = th22 - th21
th0 = th0 * 2048 * ngear / 360: th1 = th1 * 2048 * ngear / 360: th2 = th2 * 2048 * ngear / 360
a = Form1.B_8164.SetPosition(0, 0): a = Form1.B_8164.SetPosition(1, 0): a =
Form1.B_8164.SetPosition(2, 0)
ax(0) = 0: ax(1) = 1: ax(2) = 2
a = Form1.B_8164.StartTRLLine3(ax, th0, th1, th2, 0, 100000, 0.01, 0.01)
End Sub

Sub move1() 'Move one axis
naxis = 0: th1 = 0: th2 = 360: dt = 1
dth = th2 - th1: dth = dth * nenc * ngear / 360
svel = 0: mvel = (dth - 3 * svel * Tacc) / (dt - 3 * Tacc): SVacc = (mvel - svel) / 3
a = Form1.B_8164.StartSRMove(naxis, dth, svel, mvel, Tacc, Tacc, SVacc, SVacc)
End Sub

Sub readp() 'Reads position (feedback)
Dim fdb0 As Double: Dim fdb1 As Double: Dim fdb2 As Double
a = Form1.B_8164.GetPosition(0, fdb0): fdb0 = fdb0 * 360 / nenc / ngear
a = Form1.B_8164.GetPosition(1, fdb1): fdb1 = fdb1 * 360 / nenc / ngear
a = Form1.B_8164.GetPosition(2, fdb2): fdb2 = fdb2 * 360 / nenc / ngear

```

```

Form1.Text1.Text = fdb0: Form1.Text5.Text = fdb1: Form1.Text6.Text = fdb2
End Sub

Sub savep()                'Saves last positions into a file
    Dim fdb0 As Double
    Dim fdb1 As Double
    Dim fdb2 As Double
    a = Form1.B_8164.GetPosition(0, fdb0): a = Form1.B_8164.GetPosition(1, fdb1): a =
Form1.B_8164.GetPosition(2, fdb2)
    Open fl0 + "lastp.txt" For Output As 1
    Print #1, fdb0, Chr(9), fdb1, Chr(9), fdb2
    Close #1
End
End Sub

Sub plot2()                'Plots curves on the screen
    yr0max = yr0(1): yr0min = yr0max: yr1max = yr1(1): yr1min = yr1max: yr2max = yr2(1):
yr2min = yr2max
    For k = 2 To ns
        If yr0min > yr0(k) Then yr0min = yr0(k): If yr0max < yr0(k) Then yr0max = yr0(k)
        If yr1min > yr1(k) Then yr1min = yr1(k): If yr1max < yr1(k) Then yr1max = yr1(k)
        If yr2min > yr2(k) Then yr2min = yr2(k): If yr2max < yr2(k) Then yr2max = yr2(k)
    Next k
    Form1.Picture1.Cls: Form1.Picture2.Cls: Form1.Picture3.Cls
    Form1.Picture1.Scale (0, 0.9 * yr0max)-(1.01 * tr(ns), 1.1 * yr0min)
    Form1.Picture2.Scale (0, 0.9 * yr1max)-(1.01 * tr(ns), 1.1 * yr1min)
    Form1.Picture3.Scale (0, 0.9 * yr2max)-(1.01 * tr(ns), 1.1 * yr2min)
    Form1.Picture1.PSet (tr(1), yr0(1)): Form1.Picture2.PSet (tr(1), yr1(1)): Form1.Picture3.PSet
(tr(1), yr2(1))
    For k = 2 To ns: Form1.Picture1.Line -(tr(k), yr0(k)): Form1.Picture2.Line -(tr(k), yr1(k)):
Form1.Picture3.Line -(tr(k), yr2(k)): Next k
    Form1.Picture1.PSet (tr(1), yrm0(1)): Form1.Picture2.PSet (tr(1), yrm1(1)):
Form1.Picture3.PSet (tr(1), yrm2(1))
    For k = 2 To ns: Form1.Picture1.Line -(tr(k), yrm0(k)): Form1.Picture2.Line -(tr(k), yrm1(k)):
Form1.Picture3.Line -(tr(k), yrm2(k)): Next k
End Sub

```