İSTANBUL TECHNICAL UNIVERSITY ★ INSTITUTE OF SCIENCE AND TECHNOLOGY

MODELLING, CONTROLLING AND REAL TIME SYSTEM IMPLEMENTATION OF A HEXAPOD PLATFORM

M.Sc. Thesis by
Cüneyt KARACA

Department : Control and Automation Engineering

Programme : Control and Automation Engineering

JANUARY 2010

İSTANBUL TECHNICAL UNIVERSITY ★ INSTITUTE OF SCIENCE AND TECHNOLOGY

MODELLING, CONTROLLING AND REAL TIME SYSTEM IMPLEMENTATION OF A HEXAPOD PLATFORM

M.Sc. Thesis by
Cüneyt KARACA
(504061108)

Date of submission : **24 December 2009**
Date of defence examination: **19 January 2010**

Supervisor (Chairman) : **Assoc. Prof. Dr. M. Turan SÖYLEMEZ (ITU)**
Members of the Examining Committee : **Prof. Dr. Hakan TEMELTAŞ (ITU)**
Prof. Dr. Ata MUĞAN (ITU)

JANUARY 2010

İSTANBUL TEKNİK ÜNİVERSİTESİ ★ FEN BİLİMLERİ ENSTİTÜSÜ

ALTI BACAKLI ROBOTİK PLATFORMUN MODELLENMESİ, KONTROLÜ VE GERÇEK ZAMANLI SİSTEM UYGULAMASI

YÜKSEK LİSANS TEZİ
Cüneyt KARACA
(504061108)

Tezin Enstitüye Verildiği Tarih : 24 Aralık 2009
Tezin Savunulduğu Tarih : 19 Ocak 2010

Tez Danışmanı : Doç. Dr. M. Turan SÖYLEMEZ (İTÜ)
Diğer Jüri Üyeleri : Prof. Dr. Hakan TEMELTAŞ (İTÜ)
Prof. Dr. Ata MUĞAN (İTÜ)

OCAK 2010

FOREWORD

There are a great number of people that I would like to express my gratitude and appreciation. First of all, I would like to express my deep appreciation and thanks for my supervisor, Assoc. Prof. Dr. M. Turan SÖYLEMEZ for his encouragement, support and guidance throughout this work.

My deepest thanks goes to my parents for always giving me the encouragement and support to do whatever I wanted to do during my whole life, I will always be deeply indebted.

I would like to give my special thanks to my love without whom I would not complete this work.

Finally, I should also convey my thanks to my chiefs and my colleagues at ASELSAN especially Tümer DOĞAN, Bektaş ARALIOĞLU, Eren BALCI and Ali YETİM for their help, support and interest.

January 2010

Cüneyt KARACA

Control Engineer

TABLE OF CONTENTS

Page

ABBREVIATIONS

MGEO	**:** Microelectronics, Guidance Electro-Optics
PCBs	**:** Printed Circuit Board
IR	**:** Infrared
DOFs	**:** Degrees of freedom
CAN	**:** Controller Area Network
ASCII	**:** American Standard Code for Information Interchange
RS-232	**:** Recommended Standard 232
RT	**:** Real-time
HIL	**:** Hardware-in-the-loop
PID	**:** Proportional-integral-derivative
RPM	**:** Revolutions per minute
FFT	**:** Fast Fourier Transform
GUI	**:** Graphical User Interface
SHM	**:** Stabilized Head Mirror
PCI	**:** Peripheral component interconnect
RP	**:** Reference Profile

LIST OF TABLES

Page

LIST OF FIGURES

Page

LIST OF SYMBOLS

l	: Number of links
j	: Number of joints
f_i	: Degrees of freedom of the ith joint
λ	: Degrees of freedom of space (3 for planar, 6 for spatial motion)
F	: Degrees of freedom of the mechanism
L_i	: The length of ith leg
U_{l_i}	: Upper leg length of the ith leg
L_{l_i}	: Lower leg length of the ith leg
r_b	: The radius of the base platform
r_m	: The radius of the top platform
α	: The angle between leg pairs
x_b, y_b, z_b	: The base platform coordinate system
x_m, y_m, z_m	: The top platform coordinate system
β_i	: The angle defines the orientation of the universal joint of L_i from x_b about z_b
λ_i	: The angle defines the orientation of the platform gimbal joint on L_i from x_m about z_m
θ_{1i}, θ_{2i}	: Define the angles of the universal joint about its axis z_{1i} and z_{2i}, respectively
θ_{3i}	: Defines the angle of the revolute joint about z_{3i} axis
ρ	: The angle defines the orientation of L_{l_i} about the base platform
l_i	: The vector between the centers of the universal joint at the base and the gimbal joint at the top platform for the ith leg
H_p	: The vector between the centers of the base and the top platform
F_{pi}	: The vector between the center of the base platform and leg's base
${}^{mobile}M_{pi}$	: The vector defines the position of the center of the gimbal joint respect to top frame
M_{pi}	: The vector defines the position of the center of the gimbal joint respect to base frame
R	: The rotation matrix
a_{i-1}	: The distance from z_i to z_{i+1} along x_i
α_{i-1}	: The angle between z_i to z_{i+1} about x_i
d_{i-1}	: The distance from x_{i-1} to x_i along z_i

$\boldsymbol{\theta_i}$	**:** The angle between x_{i-1} to x_i about z_i
$\boldsymbol{T}$	**:** The transformation matrix
$\boldsymbol{\theta_{error\,i}}$	**:** The motion error of the i^{th} motor
$\boldsymbol{F_i}$	**:** The applied force to the i^{th} actuator
$\boldsymbol{G(s)}$	**:** The transfer function between $\theta(s)$ (the motor angular position) and $V(s)$ (the analog speed voltage)
$\mathbf{w_n}$	**:** Natural frequency of second order transfer function
ζ	**:** Damping ratio of second order transfer function
$\boldsymbol{T_s}$	**:** The settling time
Kp	**:** The proportional gain
Ki	**:** The integral gain
Kd	**:** The derivative gain

MODELLING, CONTROLLING AND REALTIME SYSTEM IMPLEMENTATION OF A HEXAPOD PLATFORM

SUMMARY

In this study, modelling, controlling and real-time system implementation of a hexapod parallel manipulator designed as a motion simulator are explained. The designed hexapod platform has a mobile top platform, a fixed base platform and six rotationally extensible legs connecting the base platform to the top platform. The designed system has high payload capacity because of the parallel structure of the system and the reductor between upper and lower legs. A military tank's motion profile is realized using this motion simulator with six degrees of freedom. This system is used to test two-axis gyro stabilized head mirror designed in ASELSAN Inc. MGEO organization. The motion profile is obtained from the motion data of a real tank on APG track. The tank turret has two degrees of freedom. Therefore, the motions in yaw and pitch axes of the motion simulator are sufficient for the simulation of the tank motion. The designed motion simulator is an example for rapid prototyping and hardware-in-the-loop test applications. Turning a general-purpose personal computer in the system into a real-time embedded control system by using convenient software and hardware is explained in this study. A test system includes autocollimator and digital controller is designed to evaluate the stabilization performance of the head mirror. The user interfaces of the Hexapod platform control and the stabilized head mirror test software are developed in compliance with the design specifications. The designed system is approved by Design Quality Department of ASELSAN after analyzing gathered data from accelerometers on the top platform's motion axes. System modelling and application software are developed in MATLAB® 7.5.0 and LabWindows/CVI 8.0.1 programming environments. Pro Engineer Wildfire 3.0 software is used for mechanical design of the system. Modelling the Hexapod platform is done by using MATLAB® SimMechanics™, the control algorithm is developed by using MATLAB® Simulink®, real-time application software is done by using MATLAB® xPC Target Embedded Option, and the user interfaces are developed by using LabWindows/CVI.

ALTI BACAKLI ROBOTİK PLATFORMUN MODELLENMESİ, KONTROLÜ VE GERÇEK ZAMANLI SİSTEM UYGULAMASI

ÖZET

Bu çalışmada, hareket benzetim platformu olarak tasarlanan altı bacaklı bir paralel manipülatörün modellenmesi, kontrolü ve gerçek zamanlı sistem uygulaması ele alınmıştır. Tasarlanan sistem hareketli üst platform, sabit alt platform ve bu iki platformu birbirine bağlayan, dönel hareket kabiliyeti ile uzunluğu değişebilen, altı bacaktan oluşur. Sistem, paralel yapısından ve redüktör kullanımından dolayı yüksek yük taşıma kapasitesine sahiptir. Altı serbestlik derecesinde, ASELSAN A.Ş. MGEO bünyesinde, tasarlanan hareket benzetim platformuna hareket profili olarak tank hareket profili uygulanmış ve tanklarda yaygın olarak kullanılan stabilize alın aynasının test kriterleri oluşturulmuştur. Hareket profili, APG parkuru üzerindeki tankın hareket verisinden elde edilmiştir. Tank kulesi iki serbestlik derecesine sahiptir. Bu nedenle, hareket benzetim platformunun yan-sapma ve yükseliş-yunuslama eksenlerindeki hareketi, tank hareketinin benzetiminde yeterli olacaktır. Tasarlanan hareket benzetim platformu, hızlı prototipleme (rapid prototyping) ve çevrimsel donanım benzetimi (hardware-in-the-loop) test uygulamaları için örnek teşkil eder. Sistem içerisindeki genel bir bilgisayar, uygun yazılım ve donanım kullanılarak sistemin gerçek zamanlı gömülü kontrol sistemine dönüştürülmesi konusu bu çalışma kapsamında açıklanmıştır. Alın aynasının stabilizasyon performansını ölçebilmek için otokolimator ve sayısal kontrolöre sahip bir test sistemi tasarlanmıştır. Hareket benzetim platformu kontrol ve stabilize alın aynası test yazılımı kullanıcı arayüzleri tasarım gerekliliklerine göre geliştirilmiştir. Üst platform hareket eksenleri üzerindeki ivmeölçerlerden toplanan veriler incelenerek, tasarlanan sistem ASELSAN MGEO Tasarım Kalite Bölümü tarafından doğrulanmıştır. Sistem modelleme ve uygulama yazılımları MATLAB® 7.5.0 ve LabWindows/CVI 8.0.1 ortamlarında geliştirilmiştir. Mekanik tasarımda ise Pro Engineer Wildfire 3.0 programı kullanılmıştır. Platformun modellenmesi MATLAB® SimMechanics™ kullanılarak, kontrol algoritması MATLAB® Simulink® kullanılarak, gerçek zamanlı sistem yazılımı MATLAB® xPC Target Embedded Option kullanılarak, kullanıcı arayüzleri LabWindows/CVI kullanılarak yapılmıştır.

1. INTRODUCTION

Robotics, technology dealing with the construction and use of robots, services for needs of all human beings in manufacturing, assembly and packing, transport, earth and space exploration, surgery, defence, weaponry, simulation research, laboratory research, safety, and mass production of consumer and industrial goods. With developments in electronics, computing and robotics area, the complicated design issues are solved easily.

Several design, implementation and testing stages are needed during developing a dynamic system in large scope. Eligible design and implementation of a system can be done only after a successful testing procedure. Therefore, performance testing of the designed prototype is a critical issue. If the dynamic system is going to be used in a moving vehicle, the motion of the vehicle should be performed or simulated to see the effects on the dynamic system.

ASELSAN develops new generation stabilized head mirror for tank turrets. Performance tests of the stabilized head mirror should be done. These performance tests should be originated from a tank motion. Different ways are available to carry fulfillment of the tank motion into laboratory environment.

The common point of all fulfillments of the tank motion is a motion simulator that performs dynamic movements of the tank to observe the effects of these motions on the system. In our case, tank motion on APG track in all six axes is recorded by sensors and this motion is simulated in laboratory environment. This study explains designed Hexapod platform that enables motion simulation in all six axes (x-axis, y-axis, z-axis, roll, pitch and yaw). In addition, designed user interfaces and additional test equipments enrich the study.

This study is a good sample for implementation of hardware in the loop simulation. MATLAB xPC Target facilitates embedded control system design by turning general-purpose personal computer hardware into a rapid prototyping platform.

Labwindows/CVI and MathWorks tools such as MATLAB, Simulink, xPC Target are effectively used throughout the design cycle.

1.1 Parallel and Serial Manipulators

Serial, parallel or hybrid (parallel-serial) manipulators are used in the industry according the system requirements.

Serial manipulators are by far the most common industrial robots. Often they have an anthropomorphic mechanical arm structure. Their main advantage is their large workspace with respect to their own volume and occupied floor space.

Their main disadvantages are the low stiffness inherent to an open kinematic structure, errors are accumulated and amplified from link to link, the fact that they have to carry and move the large weight of most of the actuators, and the relatively low effective load that they can manipulate [1].

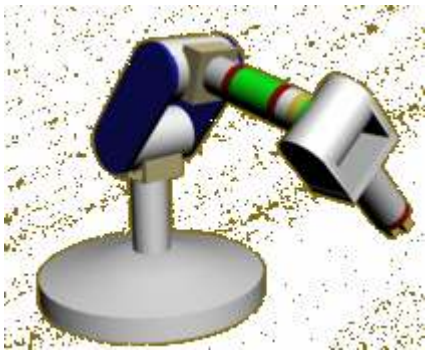

Figure 1.1 : An example of a serial manipulator.

A generalized parallel manipulator is a closed-loop kinematic chain mechanism whose end-effector is linked to the base by several independent kinematic chains. A parallel manipulator, consists of a fixed "base" platform, connected to an end-effector platform by means of a number of "legs". These legs often consist of an actuated prismatic joint, connected to the platforms through passive (i.e. not actuated) spherical and/or universal joints. Hence, the links feel only traction or compression, not bending, which increases their position accuracy and allows a lighter construction. Parallel manipulators have (in principle) high structural stiffness, since the end-effector is supported in several places at the same time. All these features result in manipulators with a wide range of motion capability. Their major drawback is their limited workspace, because the legs can collide. Major

industrial applications of these devices are airplane simulators, automobile, simulators, and in work processes. They also become more popular in high speed, high-accuracy positioning with limited workspace, such as in assembly of PCBs, as micromanipulators mounted on the end-effector of larger but slower serial manipulators, and as high speed/high-precision milling machines.

Parallel manipulators are usually faster than traditional articulated robots, since the motors can be mounted on the base, thus saving weight. They are also stronger than serial manipulators because the end-effector is connected to more links. Another benefit is that the error of the end-effector is less than the errors of serial manipulators since the errors are averaged (as opposed to being additive as in serial manipulators). However, parallel manipulators are usually more limited in the workspace; for instance, they generally cannot reach around obstacles. The calculations involved in performing a desired manipulation (forward kinematics) are also usually harder and have more than one solution. The Stewart platform and the Delta robot are popular examples [2].

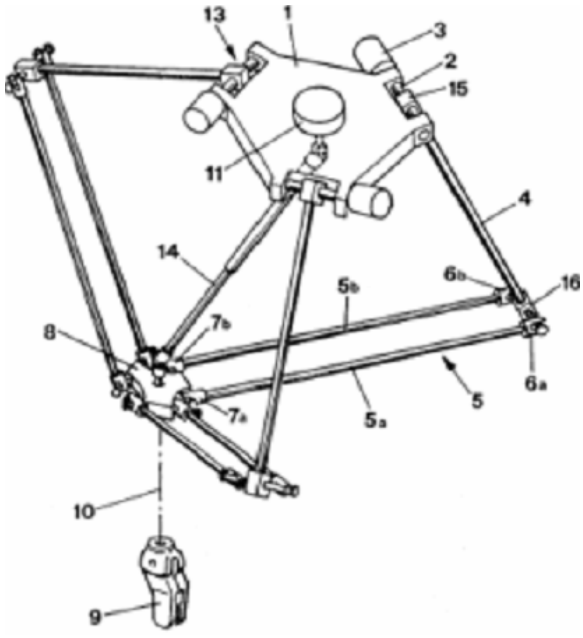

Figure 1.2 : Technical drawing of a Delta robot.

Stewart presented the Stewart platform to academia in 1965 as a flight simulator [3].

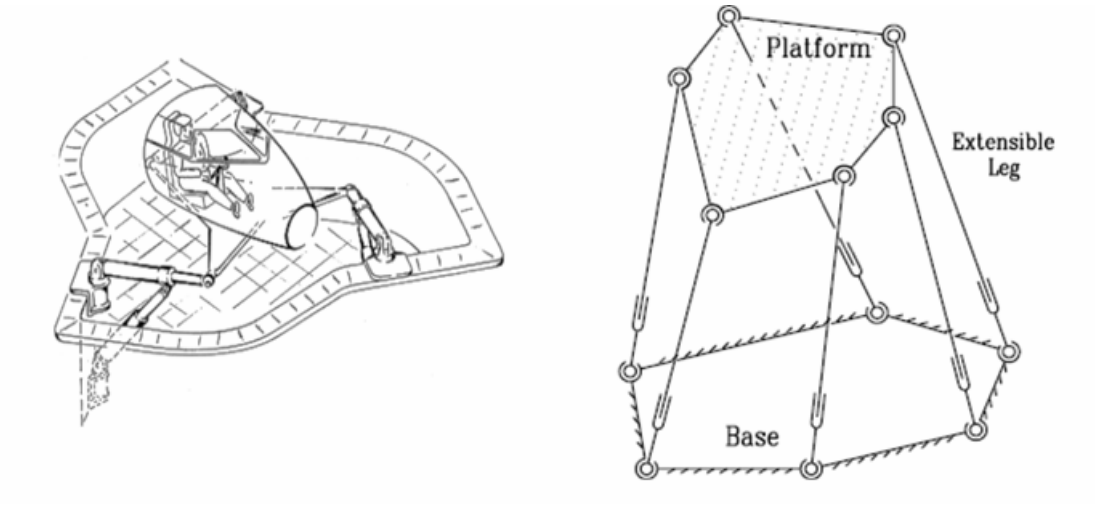

Figure 1.3 : Schematic of the Stewart platform.

Hybrid type manipulators are a combination of closed-chain and open-chain mechanisms or it is a sequence of parallel mechanisms. Parallel manipulators have high stiffness and large load capacity, but suffer from very limited workspace volume. The hybrid manipulators take advantages of parallel and serial mechanisms, and overcome the limited workspace of the parallel manipulators, and have high stiffness at the same time [4].

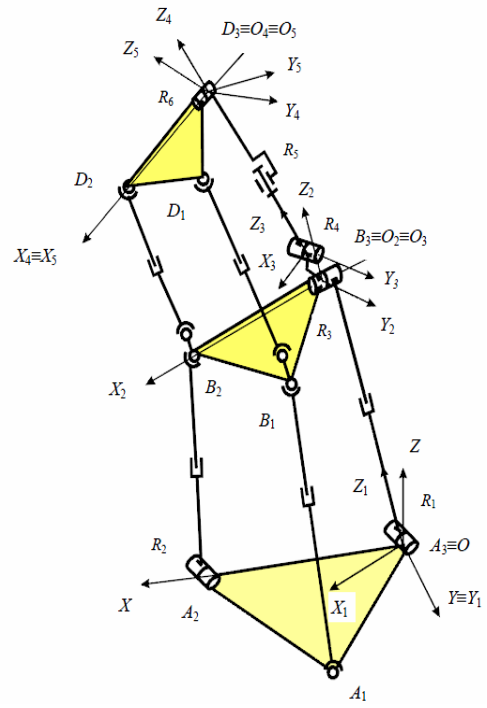

Figure 1.4 : A hybrid manipulator.

1.2 System Description

Day and IR (infrared) cameras are widely used on contemporary tanks. The main purpose of the all viewing capabilities is to produce excellent vision without any disturbances. One of the most important parts to produce stabilized vision is the compensation of the vibration produced during the tank motion. ASELSAN develops stabilized head mirrors to serve this purpose. Head mirror on the tank turret compensates the undesired shakes on the image while the tank is moving and produces clean and stabilized vision for the gunner.

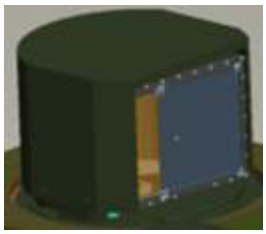

Figure 1.5 : Stabilized head mirror.

A test system is required to test the stabilization performance of the head mirror. Designing a motion simulator named as Hexapod platform is the main labor of the test set-up. The Hexapod platform simulates the tank motion by means of rotational and translational movements. Flight dynamics parameters (x-axis, y-axis, z-axis, roll, pitch and yaw) can define the test motion profile.

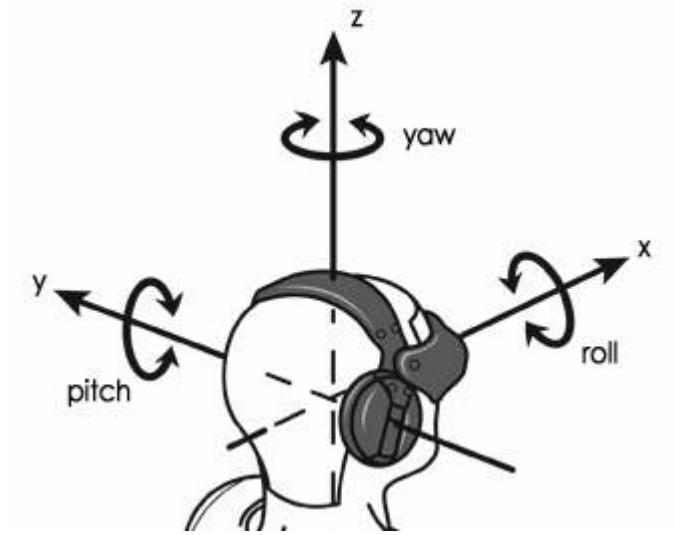

Figure 1.6 : Motion terminology.

The Hexapod platform is composed of a fixed and a mobile platform connected by six legs and can perform 6-DOF motion. The designed mechanism is shown in the following figure.

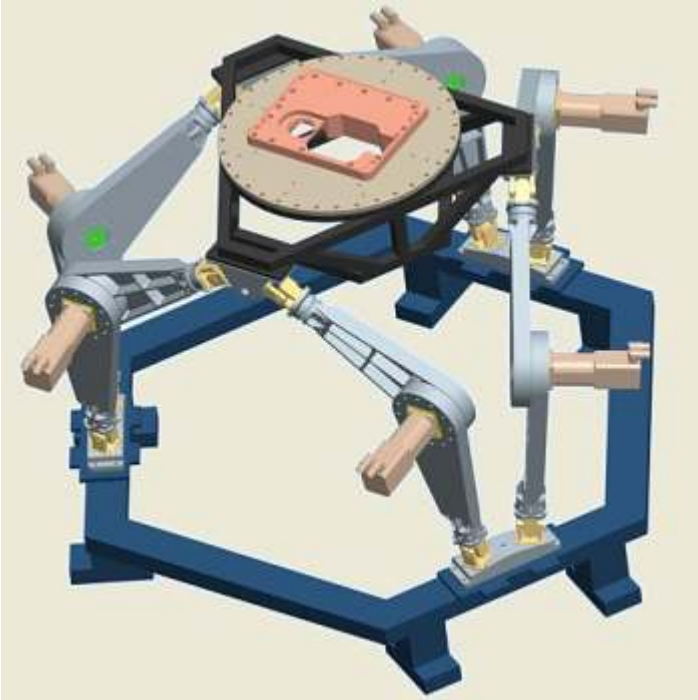

Figure 1.7 : The Hexapod mechanism.

Autocollimator, task and user interface software are added to the test system to measure the stabilization performance of the head mirror. The following figure shows the designed test system for the performance test of the stabilized head mirror.

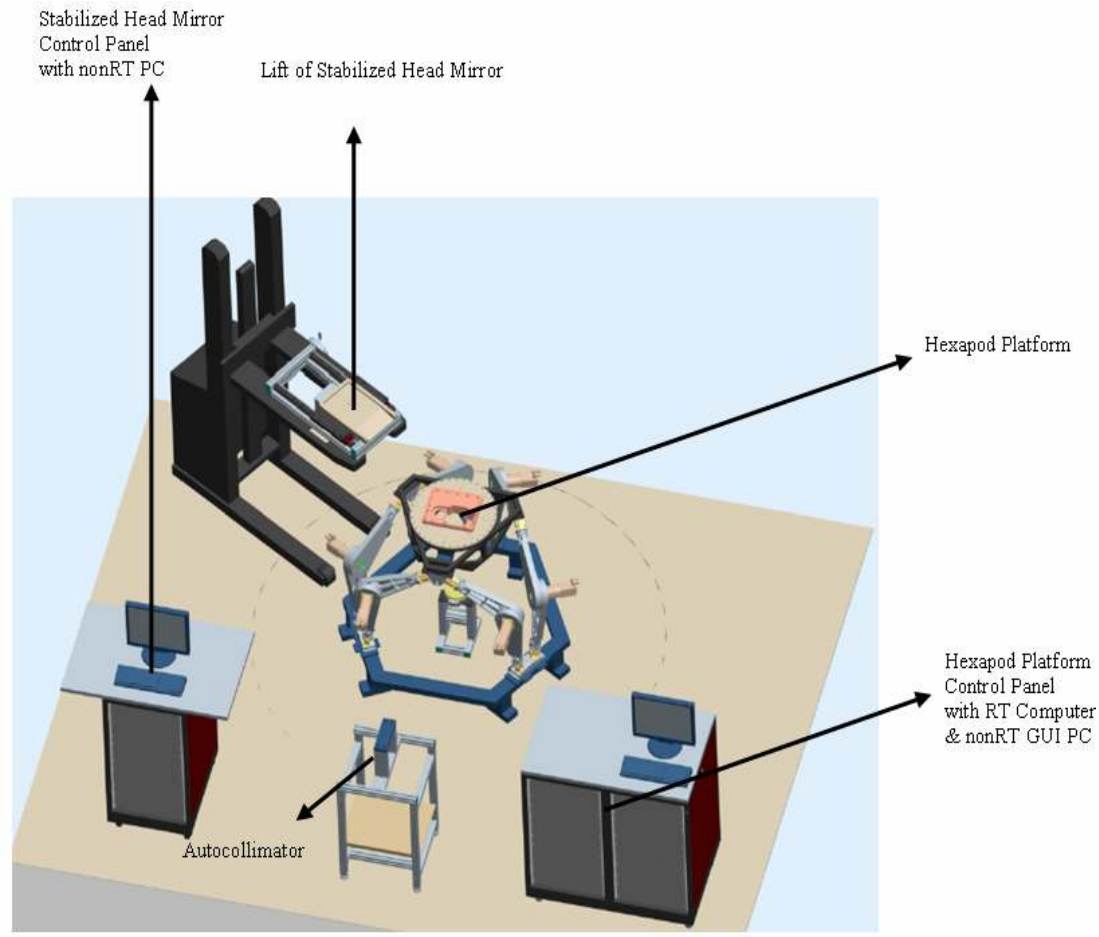

Figure 1.8 : Designed test system for the stabilized head mirror.

1.3 Organization of Thesis

The study is organized as follows. In Section 2, kinematics of the Hexapod platform including characteristics of the mechanism, inverse and forward kinematics models are elaborated. Section 3 gives an overview of how modelling is used in the system design and describes the control strategy as used for the Hexapod platform. Section 4 discusses the concept of rapid prototyping, hardware in the loop simulation and embedded control system. Real-time system implementation is presented with all the specifics that include electrical design, the system components such as actuators, resolvers, reductors, I/O cards, hardware and software implementations, motion profile generation, controller implementation, real-time computing and embedded code generation, design of graphical user interfaces. Verification of the motion simulator (the Hexapod platform) is discussed in the scope of this section. In addition, this section explains how to test the performance of the stabilized head mirror. Finally, section 5 presents conclusions.

MATLAB codes, specifications of the system elements, and technical data are presented in Appendices.

2. KINEMATICS OF THE HEXAPOD PLATFORM

Kinematics is the representation of a mechanism's motion without taking into count forces, torques and the mass properties of the mechanism. The mechanism's motion can be integrated, if forces, torques, mass properties of the mechanism, the initial positions and their velocities are known. Kinematics analysis of the mechanism is based on two kinematics problems, which are forward kinematics and inverse kinematics. Forward kinematics concerns with computation of the position and the orientation of the mechanism's tip point as a function of the mechanism's joint angles or lengths. On the contrary, inverse kinematics concerns with determination of the mechanism's joint angles or lengths in order to set the mechanism desired position and orientation. In general, inverse kinematics solution is easier than forward kinematics solution for parallel manipulators.

2.1 Characteristics of the Hexapod Platform

The hexapod platform can be easily and precisely positioned and oriented. Also, it can easily follow a reference trajectory in its workspace.

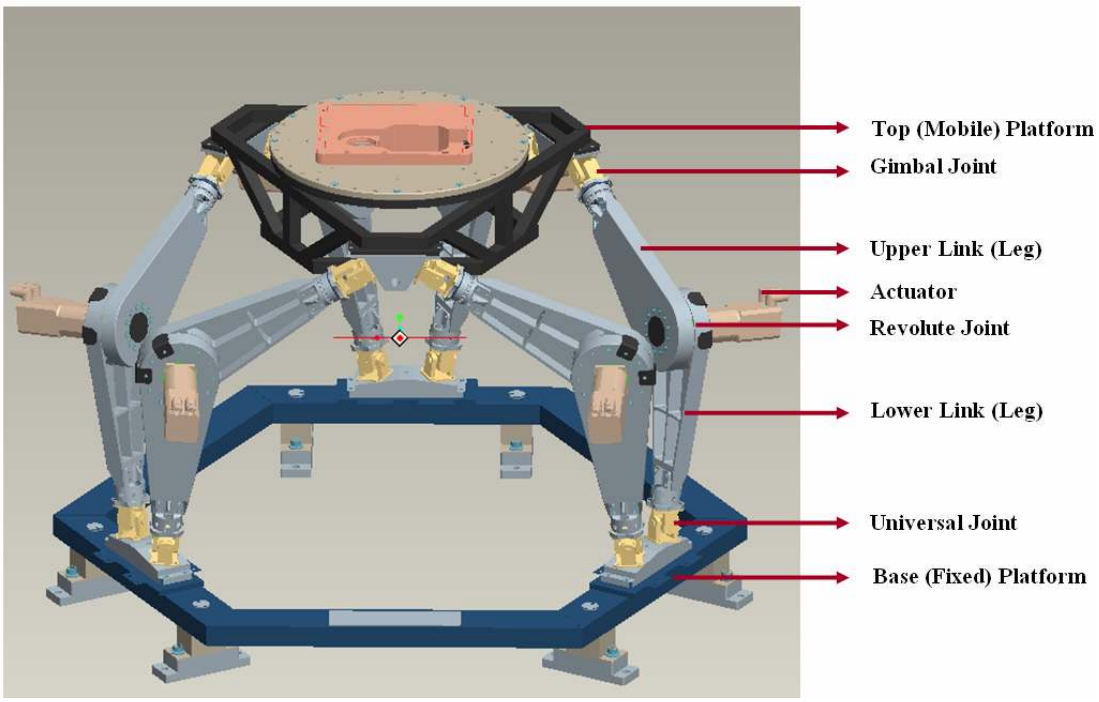

Figure 2.1 : The Hexapod platform's mechanism description.

The Hexapod Platform shown in Figure 2.1 is quite complex due to mechanical constraints. The Hexapod Platform consists of a fixed platform and a mobile platform, which are connected to each other by six legs. The actuators are placed at the intersection of the lower and upper legs, so it acts as revolute joint. The gimbal joints are placed at the connection point of the upper legs and the top platform. Finally, the universal joints are placed at the connection point of the lower legs and the base platform. The whole assembly is a parallel mechanism, which has six Cartesian degrees of freedom (DOF). The desired position or orientation of the Hexapod Platform is accomplished by associating with the six legs' angles, transforming the six transitional DOF into three positional (x, y, and z) and three orientational ones (roll, pitch and yaw).

2.1.1 Analyzing degrees of freedom

Degrees of freedom (DOF) are the set of independent displacements or rotations that define the displaced position and orientation of the mechanism. The designed Hexapod Platform has six Cartesian degrees of freedom. It can be confirmed by common degrees of freedom equations. The assembly of the Hexapod Platform contains a gimbal joint which has three rotational primitives (3 DOFs) between mobile platform and upper leg, a revolute joint which has a rotational primitive (1

DOF) between upper leg and lower leg, a universal joint which has two rotational primitives (2 DOFs) between fixed platform and lower leg for each legs. Both the mobile and fixed platforms are defined as a link.

A general form of the degrees of freedom equation can be represented as [5],

$$F = \lambda.(l - j - 1) + \sum f_i \qquad \textbf{(2.1)}$$

For the assembly of a leg, the mechanism has two links, which are upper leg and lower leg, three joints, which are universal, revolute and gimbal joints and six joint DOFs that are six rotational primitives. Therefore, l (number of links) = 2, j (number of joints) = 3 and f_i (number of joint DOFs) = 6 are obtained.

For the full assembly of the Hexapod Platform, the mechanism has six legs. Therefore, total number of links = 6.l + 2 (mobile platform and fixed platform are counted as link) = 14, total number of joints = 6.j = 18, total number of joint DOFs = 6. f_i = 36.

In equation (2.1) λ represents space DOFs. Therefore, λ equals to six for spatial space.

Counting degrees of freedom of the Hexapod Platform can be done by calculating F in equation (2.1). F = 6×(14 – 18 – 1) + 36 = 6. Therefore, the Hexapod Platform has six independent DOFs (x-axis, y-axis, z-axis, roll, pitch and yaw).

2.2 Inverse Kinematics of the Hexapod Platform

It is critical to explain the terminology before analyzing kinematics details. Figure 2.2 shows the terminology of the Hexapod Platform to describe the specifics. The Hexapod Platform has a mobile and a fixed platform that are connected with six legs (L_i). Each leg has two link lengths which are upper leg length (U_{l_i}) and lower leg length (L_{l_i}), a universal joint at the bottom, a revolute joint at the elbow and a gimbal joint at the top. A base frame and a mobile platform frame are placed at the center of the fixed platform and the mobile platform respectively. r_b is the radius of the fixed platform that it defines the distance between the lower leg mounting location and the center of the base platform and r_m is the radius of the mobile platform that it defines

the distance between the upper leg mounting location and the center of the mobile platform. α is the angle between leg pairs such as L_1-L_6, L_2-L_3 and L_4-L_5 for the base plate; L_1-L_2, L_3-L_4 and L_5-L_6 for the top plate as shown in Figure 2.3. β_i angle defines the orientation of the universal joint of L_i from x_b about z_b. λ_i angle defines the orientation of the platform gimbal joint on L_i from x_m about z_m. θ_{1i} and θ_{2i} define the angles of the universal joint about its axis z_{1i} and z_{2i}. θ_{3i} defines the angle of the revolute joint about z_{3i} axis. ρ angle defines the orientation of L_{l_i} about the base platform. The initial angles of the revolute joints are set to 120 degrees as the system startup configuration.

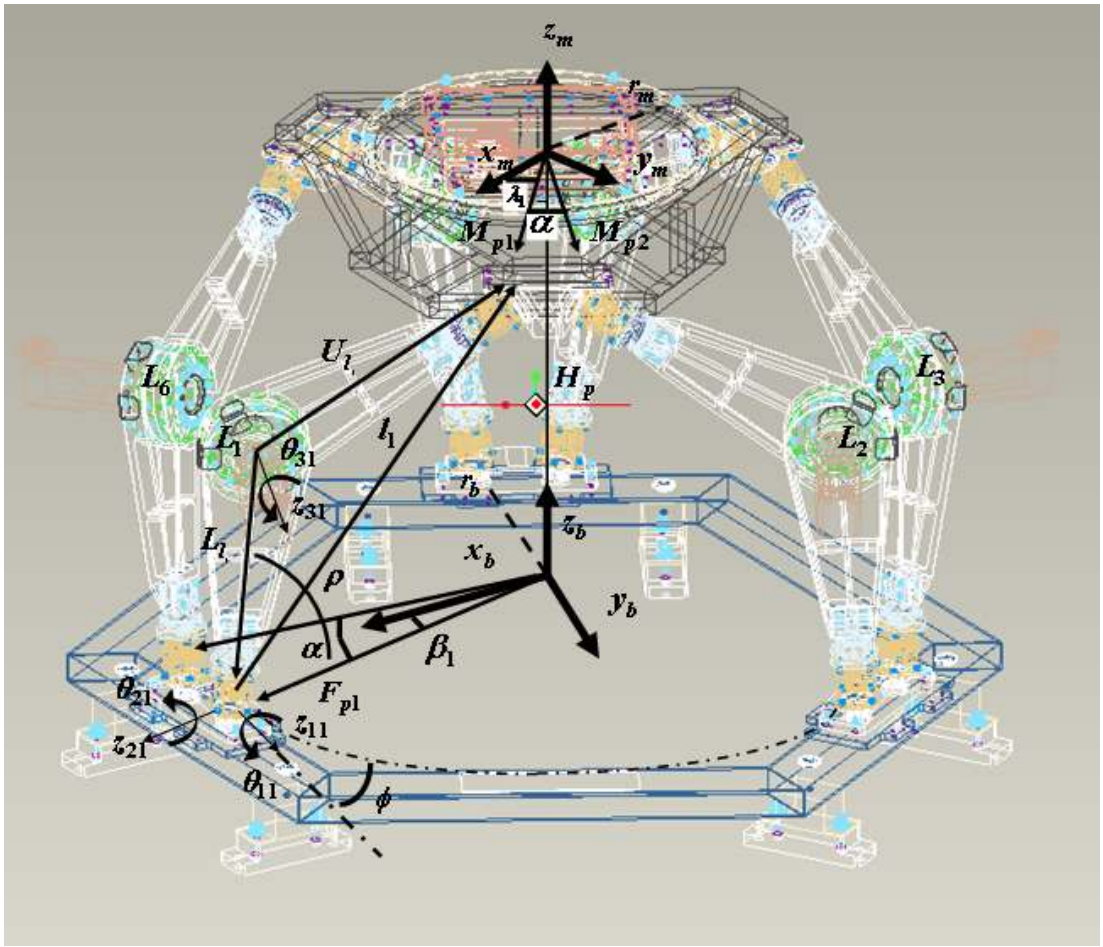

Figure 2.2 : The Hexapod platform's terminology.

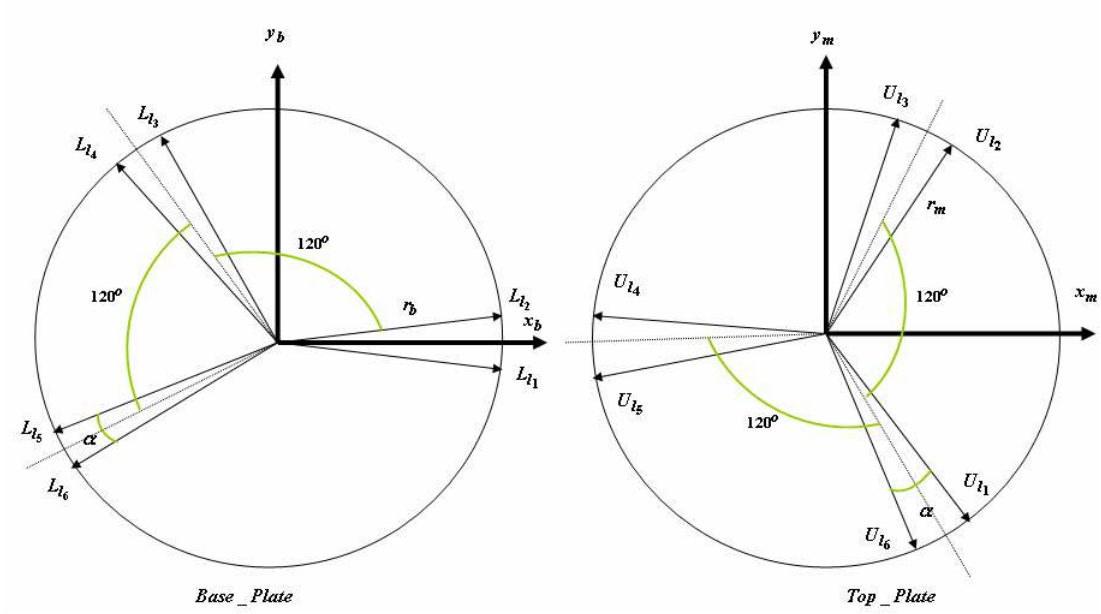

Figure 2.3 : Geometric layouts of base and top plates.

The inverse kinematics of the Hexapod Platform bases on finding the angles of the revolute joints of each leg in order to provide the desired position and orientation for the mobile platform. This problem can be solved by finding the expression for l_i vector, which is length between the universal joint and the gimbal joint of a leg. The expression for l_i includes M_{pi}, F_{pi} and H_p vectors as seen from Figure 2.2. Another expression for l_i can be written in terms of link lengths of the leg and the joint angles.

$$l_i = H_p + M_{pi} - F_{pi} \qquad \textbf{(2.2)}$$

The vector equation (2.2) can be obtained geometrically from Figure 2.2. H_p vector given in equation (2.2) has H_{px}, H_{py} and H_{pz} elements. F_{pi} defines the position of the center of the universal joint respect to the base frame. $[\, r_b \cos\beta_i , r_b \sin\beta_i , 0 \,]$ are the elements of F_{pi}. M_{pi} defines the position of the center of the gimbal joint respect to base frame. The equation (2.3) shows the calculation of M_{pi}.

$$M_{pi} = {}^{base}_{mobile}R \; {}^{mobile}M_{pi} \qquad \textbf{(2.3)}$$

$[\, r_m \cos\lambda_i , r_m \sin\lambda_i , 0\,]$ are the elements of ${}^{mobile}M_{pi}$ The rotation matrix ${}^{base}_{mobile}R$ is a three by three matrix describing the orientation of the mobile platform's coordinate frame relative to the fixed platform's coordinate frame.

The equation (2.4) is the combination of the equation (2.2) and the equation (2.3).

$$\begin{bmatrix} l_{ix} \\ l_{iy} \\ l_{iz} \end{bmatrix} = \begin{bmatrix} H_{px} \\ H_{py} \\ H_{pz} \end{bmatrix} + \begin{bmatrix} R_{11}R_{12}R_{13} \\ R_{21}R_{22}R_{23} \\ R_{31}R_{32}R_{33} \end{bmatrix} \cdot \begin{bmatrix} r_m \cos\lambda_i \\ r_m \sin\lambda_i \\ 0 \end{bmatrix} - \begin{bmatrix} r_b \cos\beta_i \\ r_b \sin\beta_i \\ 0 \end{bmatrix} \qquad \textbf{(2.4)}$$

l_i is written respect to the base platform's coordinate frame in the above equation. The transformation of l_i to the legs' coordinate frame can be written as the following expression.

$${}^{leg_i}l_i = {}^{base}_{leg_i}R^T \cdot {}^{base}l_i \qquad \textbf{(2.5)}$$

The rotation matrix ${}^{base}_{leg_i}R$ includes three rotational information. These are rotation about z_b by β_i, rotation about x' by $-(90+k\phi)$ where k is $(-1)^{i+1}$ for L_i and rotation about z' by $(\rho - 180)$.

$${}^{base}_{leg_i}R = \text{Rot}\,(z_b, \beta_i)\ \text{Rot}\,(x', -(90+k\phi))\ \text{Rot}\,(z', (\rho-180)) \qquad \textbf{(2.6)}$$

Expand the equation (2.6) and get the following equation that c and s letters denote cos and sin functions respectively.

$$\left[{}^{base}_{leg_i}R\right]^T = \begin{bmatrix} -c\beta c\rho - ks\beta_i s\phi s\rho & -s\beta c\rho + kc\beta_i s\phi s\rho & c\phi s\rho \\ c\beta_i s\rho - ks\beta_i s\phi c\rho & s\beta_i s\rho + kc\beta_i s\phi c\rho & c\phi c\rho \\ -s\beta_i c\phi & c\beta_i c\phi & -ks\phi \end{bmatrix} \qquad \textbf{(2.7)}$$

${}^{leg_i}l_i$ can be calculated by using the equation (2.7) and the equation (2.5).

Alternately, the Denavit-Hartenberg convention can be used. As seen from Figure 2.2, the modified Denavit-Hartenberg parameters can be written as the following table.

Table 2.1 : The modified Denavit-Hartenberg parameters.

$link_i$	α_{i-1} (Link Twist)	a_{i-1} (Link Length)	d_i (Joint Offset)	θ_i (Joint Angle)
1	0	0	0	θ_1
2	$-\pi/2$	0	0	θ_2
3	$\pi/2$	L_l	0	θ_3
4	0	U_l	0	θ_4

a_{i-1} is the distance from z_i to z_{i+1} along x_i.

α_{i-1} is the angle between z_i to z_{i+1} about x_i.

d_{i-1} is the distance from x_{i-1} to x_i along z_i.

θ_i is the angle between x_{i-1} to x_i about z_i.

${}^0_4T = {}^0_1T\,{}^1_2T\,{}^2_3T\,{}^3_4T$ can be written to express the position and the orientation of the reference frame positioned at the center of the gimbal joint respect to the leg frame.

$$\begin{aligned} {}^{leg_i}l_{ix} &= L_l c\theta_1 c\theta_2 + U_l c\theta_1 c\theta_2 c\theta_3 - U_l s\theta_1 s\theta_3 \\ {}^{leg_i}l_{iy} &= L_l s\theta_1 c\theta_2 + U_l s\theta_1 c\theta_2 c\theta_3 + U_l c\theta_1 s\theta_3 \\ {}^{leg_i}l_{iz} &= -L_l s\theta_2 - U_l s\theta_2 c\theta_3 \end{aligned} \qquad \textbf{(2.8)}$$

The angle θ_3 can be calculated from the equation (2.8) as the following equation.

$$\theta_3 = \arccos\left(\frac{{}^{leg_i}l_{ix}{}^2 + {}^{leg_i}l_{iy}{}^2 + {}^{leg_i}l_{iz}{}^2 - L_l{}^2 - U_l{}^2}{2L_l U_l}\right) \qquad \textbf{(2.9)}$$

The angle θ_3 can also be evaluated from cosine theorem.

$${}^{leg_i}l_i{}^2 = {}^{leg_i}l_{ix}{}^2 + {}^{leg_i}l_{iy}{}^2 + {}^{leg_i}l_{iz}{}^2 = L_l{}^2 + U_l{}^2 - 2L_l U_l \cos(\pi + \theta_3) \qquad \textbf{(2.10)}$$

$${}^{leg_i}l_{ix}{}^2 + {}^{leg_i}l_{iy}{}^2 + {}^{leg_i}l_{iz}{}^2 = L_l{}^2 + U_l{}^2 + 2L_l U_l \cos\theta_3 \qquad \textbf{(2.11)}$$

Then, the angle θ_3 can be written as the equation (2.9).

Additionally, the angle θ_2 and the angle θ_1 can be calculated as the following equations.

$$\theta_2 = \arcsin\left(\frac{-{}^{leg_i}l_{iz}}{U_l c\theta_3 + L_l}\right) \quad \textbf{(2.12)}$$

$$\theta_1 = 2\arctan\left(\frac{-U_l s\theta_3 \pm (U_l{}^2 s^2\theta_3 + c^2\theta_2 (L_1 + U_l c\theta_3)^2 - {}^{leg_i}l_{ix}{}^2)^{1/2}}{U_l c\theta_3 c\theta_2 + L_l c\theta_2 + {}^{leg_i}l_{ix}}\right) \quad \textbf{(2.13)}$$

2.2.1 Inverse kinematics model in MATLAB, Simulink

In this study, simulation and software implementation are achieved by using MATLAB that is a high-level programming language and interactive environment that enables you to perform computationally intensive tasks [6]. Simulink is a powerful development tool of MATLAB that it is an environment for multi domain simulation and model-based design for dynamic and embedded systems. It provides an interactive graphical environment and a customizable set of block libraries that let you design, simulate, implement, and test a variety of time-varying systems, including communications, controls, signal processing, video processing, and image processing [7]. The designed model includes the blocks of SimMechanics, which is a toolbox that extends Simulink software with tools for modelling and simulating mechanical systems. It is integrated with MathWorks™ control design and code generation products, enabling you to design controllers and test them in real time with the model of the mechanical system [8].

The inverse kinematics' parent system has two input ports (the rotational and translational degrees of freedom) and an output port which are desired Angular (roll, pitch, yaw), Cartesian (x, y, z) profile inputs and Motor Angle Demand (angle of each revolute joints) output for the subsystem as shown in Figure 2.4. The Inverse Kinematics subsystem transforms the six degrees of freedom (roll, pitch , yaw, x, y, z) into the equivalent set of six degrees of freedom expressed as the motor angles of the six platform legs as shown in Figure 2.5. The output of the subsystem is a six-vector of these motor angles.

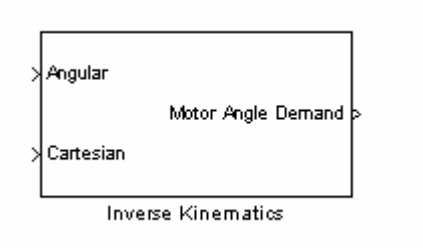

Figure 2.4 : Parent system of inverse kinematics process.

Computation of leg vectors can be achieved by the help of the equation (2.2), the equation (2.3) as shown in Figure 2.6. Roll, pitch and yaw represent rotation about X, Y and Z respectively. The rotation matrix of the model in Figure 2.6 consists of Euler angles XYZ ($R = R_x R_y R_z$).

$$R_x = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(rollAngle) & \sin(rollAngle) \\ 0 & -\sin(rollAngle) & \cos(rollAngle) \end{bmatrix} \quad \textbf{(2.14)}$$

$$R_y = \begin{bmatrix} \cos(pitchAngle) & 0 & -\sin(pitchAngle) \\ 0 & 1 & 0 \\ \sin(pitchAngle) & 0 & \cos(pitchAngle) \end{bmatrix} \quad \textbf{(2.15)}$$

$$R_z = \begin{bmatrix} \cos(yawAngle) & \sin(yawAngle) & 0 \\ -\sin(yawAngle) & \cos(yawAngle) & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad \textbf{(2.16)}$$

Finally, Euler XYZ rotation matrix (R) can be evaluated as the following equation.

$$R = \begin{bmatrix} c(yA)c(pA) & -s(yA)c(pA) & s(pA) \\ s(rA)s(pA)c(yA) + c(rA)s(yA) & c(rA)c(yA) - s(yA)s(pA)s(rA) & -s(rA)c(pA) \\ -c(rA)s(pA)c(yA) + s(rA)s(pA) & s(pA)c(rA)s(yA) + s(rA)c(yA) & c(pA)c(rA) \end{bmatrix} \quad \textbf{(2.17)}$$

In the equation (2.17) *c, s, rA, pA* and *yA* symbolize cos function, sin function, roll angle, pitch angle and yaw angle respectively.

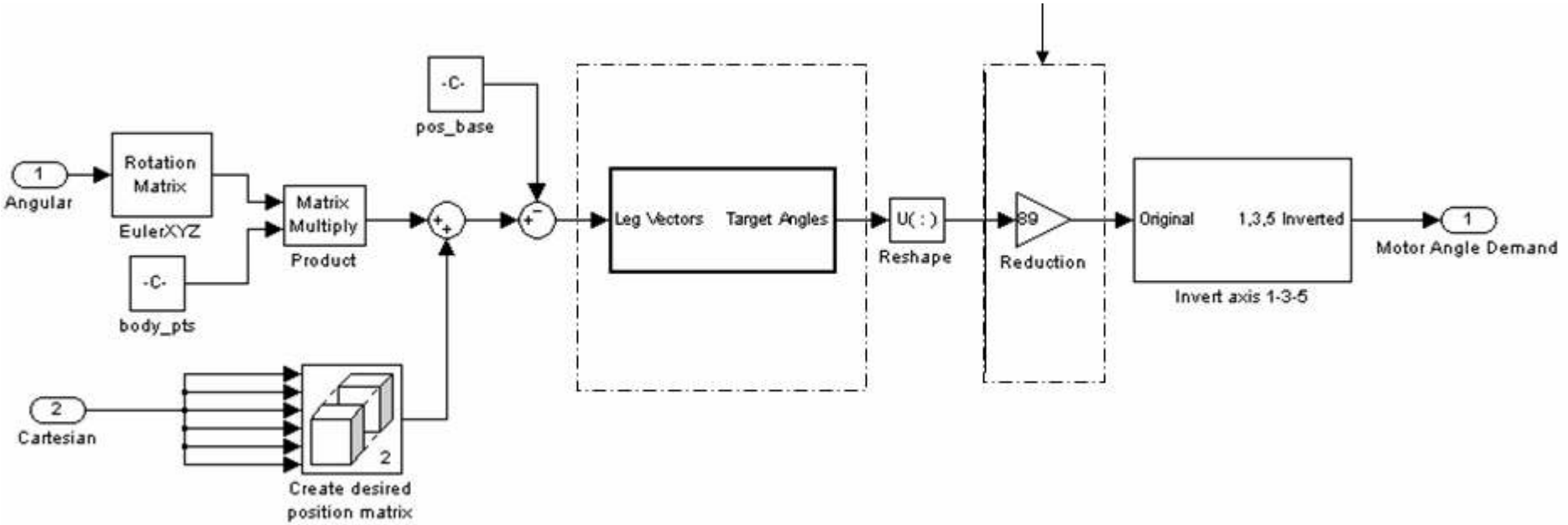

Figure 2.5 : Subsystem system of inverse kinematics process.

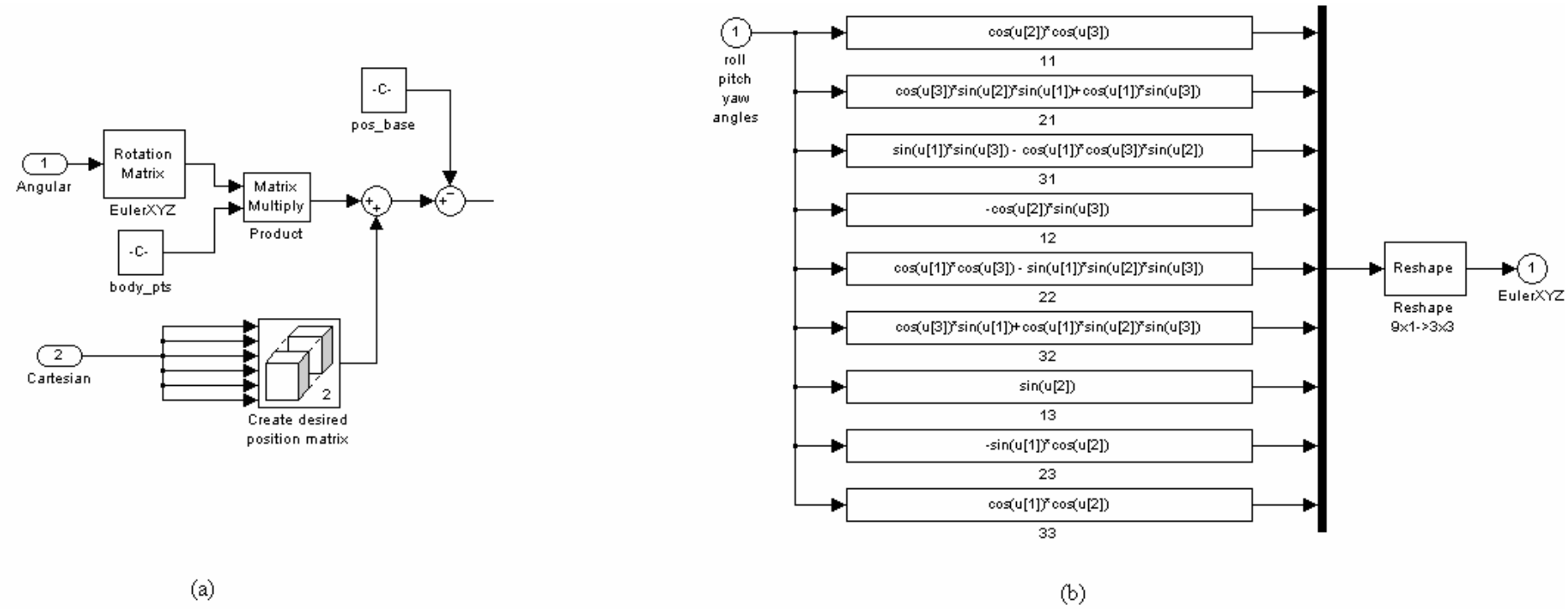

Figure 2.6 : Inverse kinematics subsystems: (a) Computation of leg vectors. (b) Euler XYZ rotation matrix subsystem.

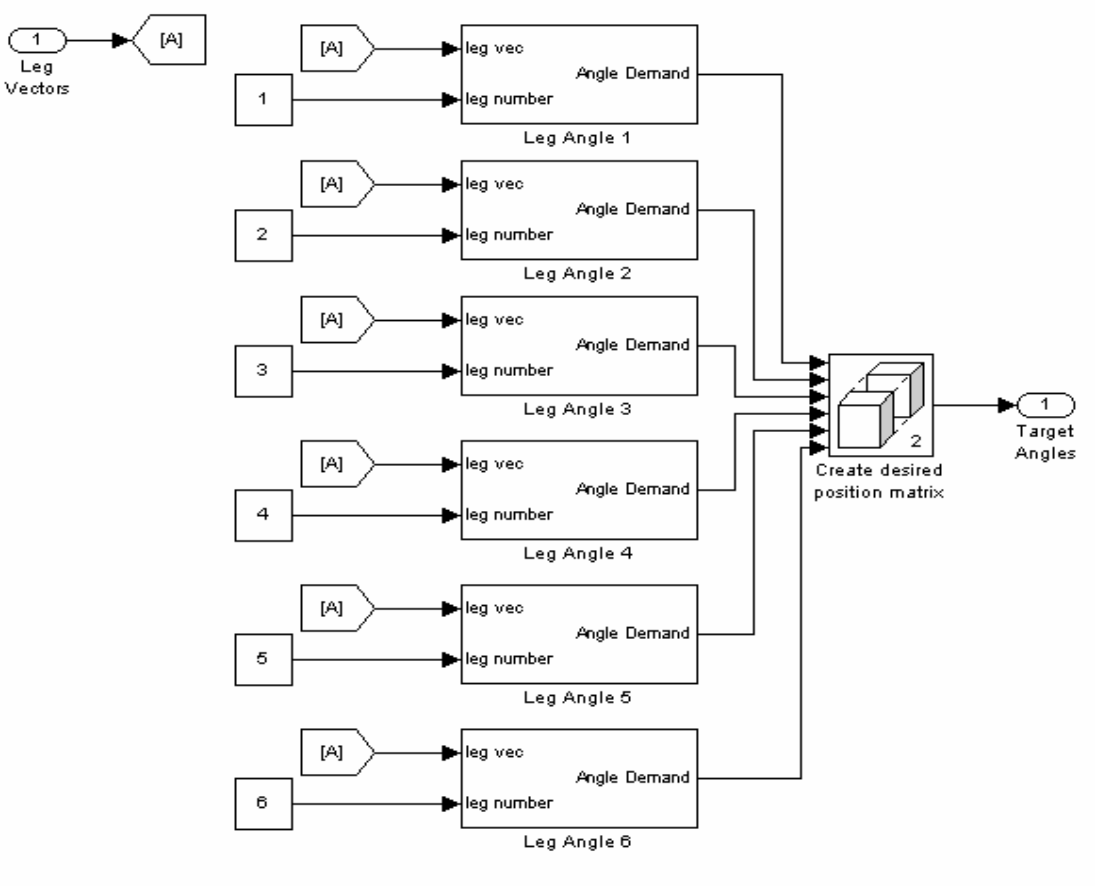

Figure 2.7 : Computation of target angles.

The following figures show the computation of the angle between upper and lower legs.

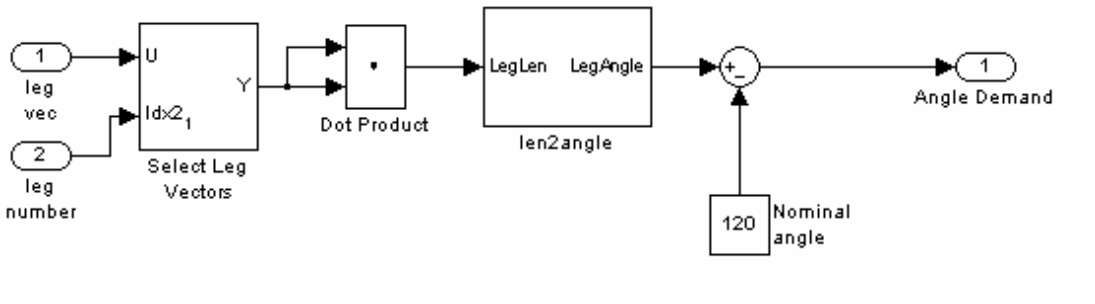

Figure 2.8 : Computation of angle demand.

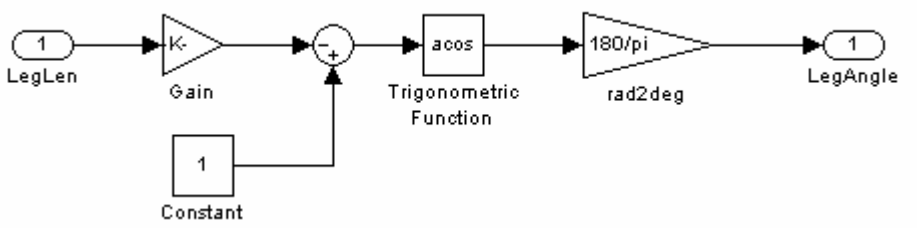

Figure 2.9 : Computation of a leg angle.

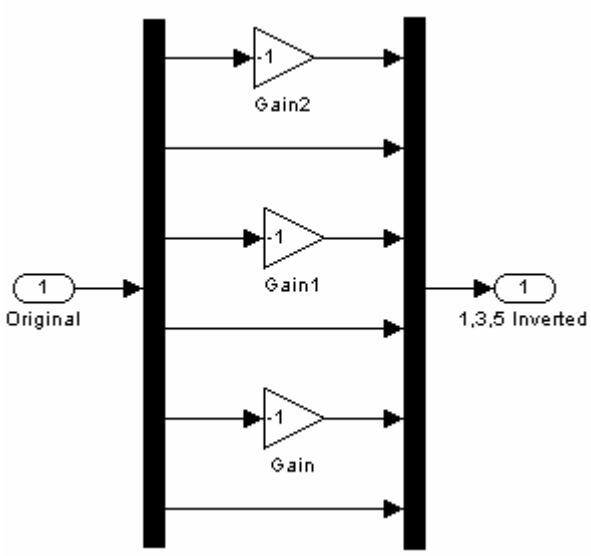

Figure 2.10 : Inversion of 1, 3, 5 leg angles.

Computation of angle demand can be achieved by the help of the equation (2.19) as shown in Figure 2.8 and Figure 2.9.

The designed Hexapod Platform has the same leg length ($U_l = L_l$ = 0.461 meters). In the equation (2.9) θ_3 is defined as the angle of the revolute joint about z_3. However, the model shown in Figure 2.8 and Figure 2.9 uses the angle named as LegAngle that equals to $(\pi + \theta_3)$.

$$LegAngle = \arccos\left(\frac{-{}^{leg_i}l_{ix}{}^2 - {}^{leg_i}l_{iy}{}^2 - {}^{leg_i}l_{iz}{}^2 + L_l{}^2 + U_l{}^2}{2L_lU_l}\right) \quad \textbf{(2.18)}$$

$U_l = L_l$, so the equation can be written in the following form.

$$LegAngle = \arccos\left(1 - \frac{{}^{leg_i}l_{ix}{}^2 + {}^{leg_i}l_{iy}{}^2 + {}^{leg_i}l_{iz}{}^2}{2L_lU_l}\right) \quad \textbf{(2.19)}$$

The reduction gear is located between the motor shaft and revolute joint. Therefore, the reduction gear factor (1/89) should be added to calculation of motor angle demand as shown in Figure 2.5.

MotorAngle = LegAngle / Reduction Gear Factor **(2.20)**

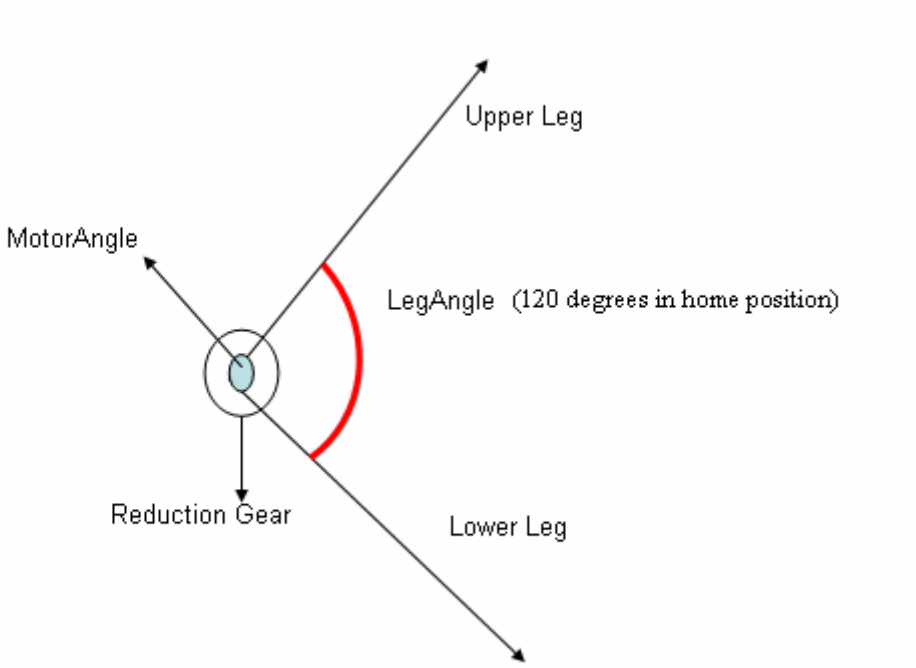

Figure 2.11 : Relation between leg angle and motor angle.

2.3 Forward Kinematics of the Hexapod Platform

The forward kinematics of the Hexapod Platform bases on finding Cartesian position and orientation of the mobile platform for the alteration of each joint angle. The forward kinematics process is composed of the inverse mapping of the inverse kinematics. This process involves the inversion of the equation (2.9) and the equation (2.2). The inversion process is hard to solve due to nonlinear equations for the unknown elements of x, y, z, roll, pitch and yaw. However, iterative techniques like Newton-Raphson can be used for the solution. In this study, the need of forward kinematics solution is to model the plant (the Hexapod Platform) in simulation. The plant can be modeled in SimMechanics. Simscape / SimMechanics library has many Simulink models such as bodies, joints, actuators, sensors etc.

2.3.1 Forward kinematics model in MATLAB, SimMechanics

In this study, simulation of the plant is achieved by using SimMechanics.

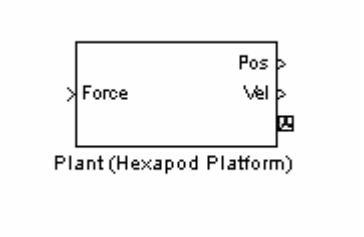

Figure 2.12 : Parent system of plant (Hexapod platform).

As shown in Figure 2.12, the model of the plant has one input port and three output ports, which are forces of the actuators, positions and velocities of revolute joints and a body position sensor of the center of gravity of the mobile platform respectively.

Figure 2.13 shows the details about plant model. The plant subsystem has six moving legs, a mobile plate and a base plate connected to the ground. CS1, CS2, CS3, CS4, CS5 and CS6 define the mobile plate's points where upper legs connected.

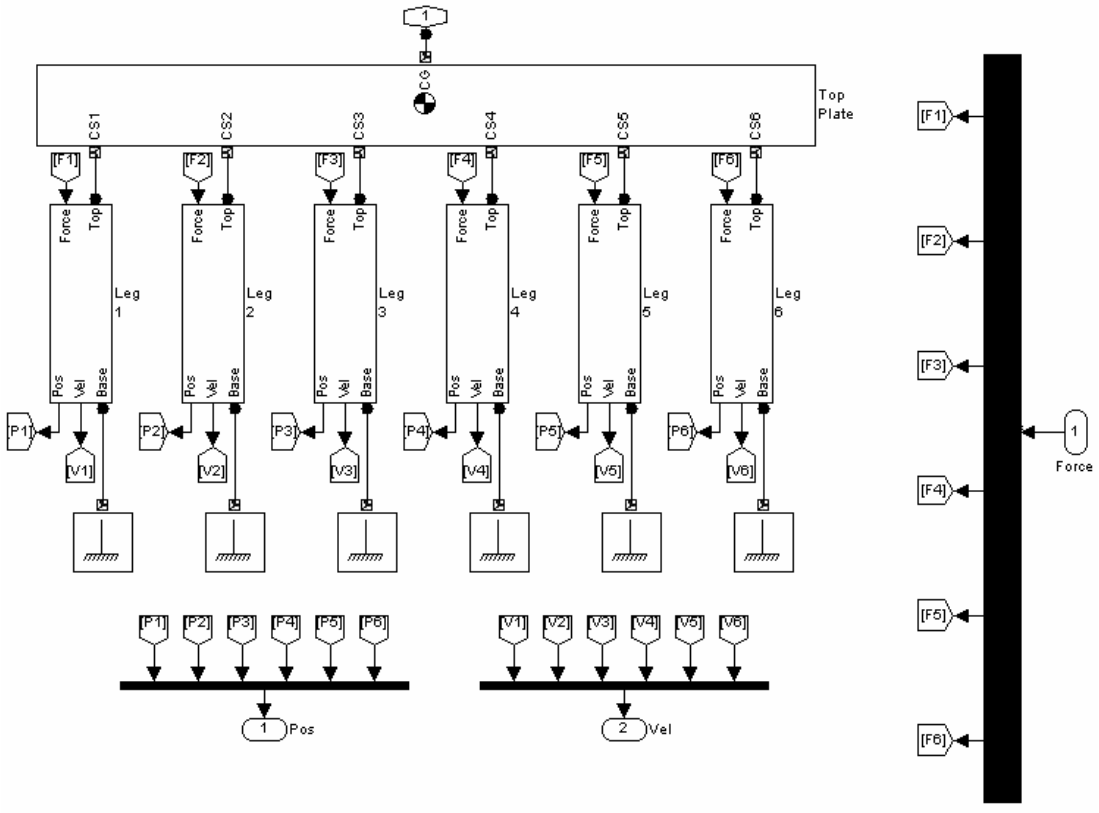

Figure 2.13 : Plant (the Hexapod platform) model in SimMechanics.

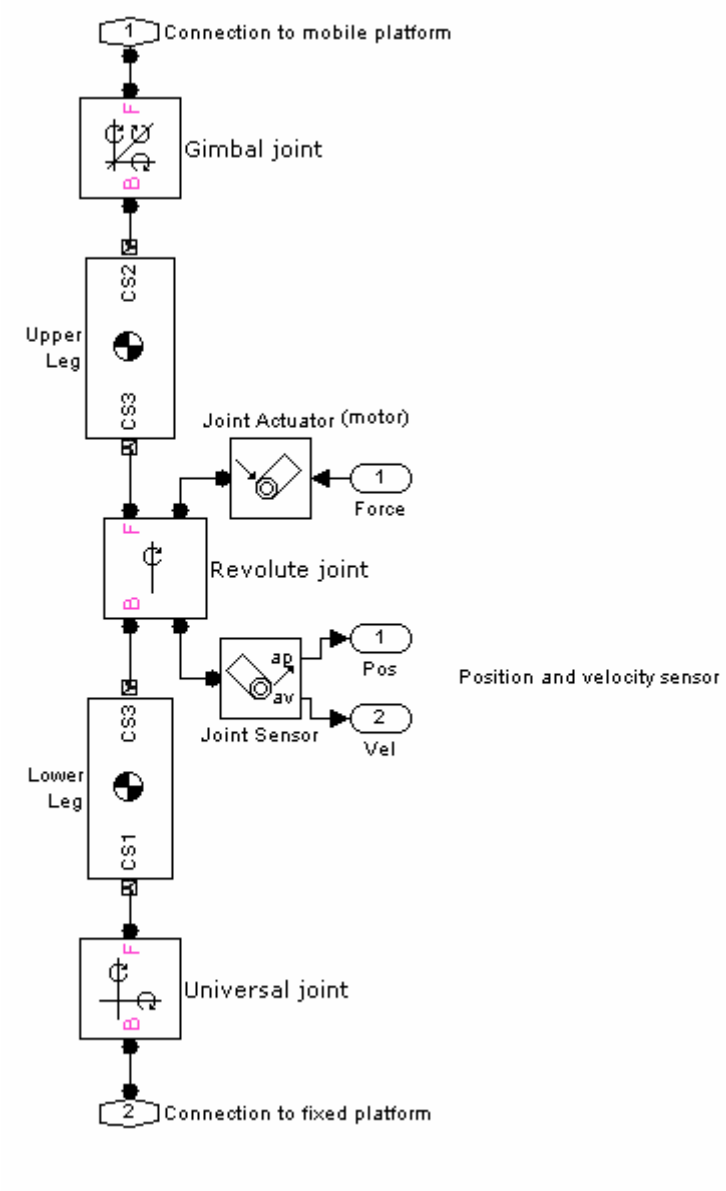

Figure 2.14 : Model of a leg in SimMechanics.

Figure 2.14 shows the model of a leg that is created based on Pro Engineer Wildfire 3.0 drawings as shown in Figure 2.15 and Figure 2.16. The lower leg is connected to fixed platform by a universal joint. The upper and lower legs are connected by a revolute joint, which is driven by joint actuator's force. A joint sensor is placed to measure angular position (ap) and angular velocity (av) of revolute joint for close loop system development and error observation. The upper leg is connected to mobile platform by a gimbal joint.

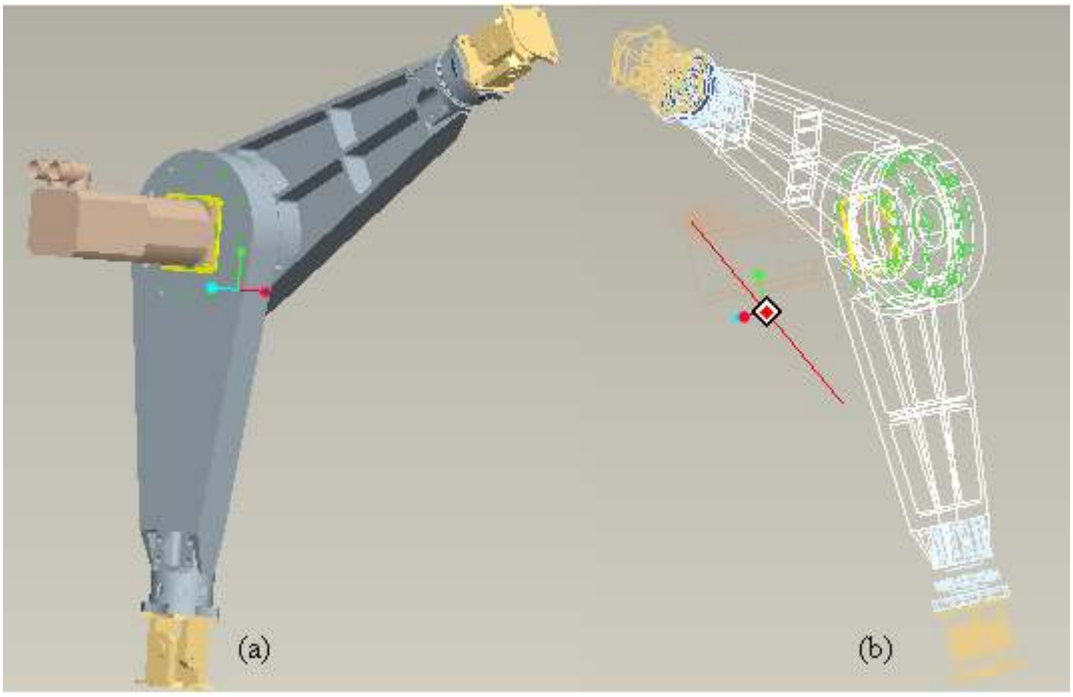

Figure 2.15 : Pro Engineer drawing of a leg: (a) Shading view. (b) Wireframe view.

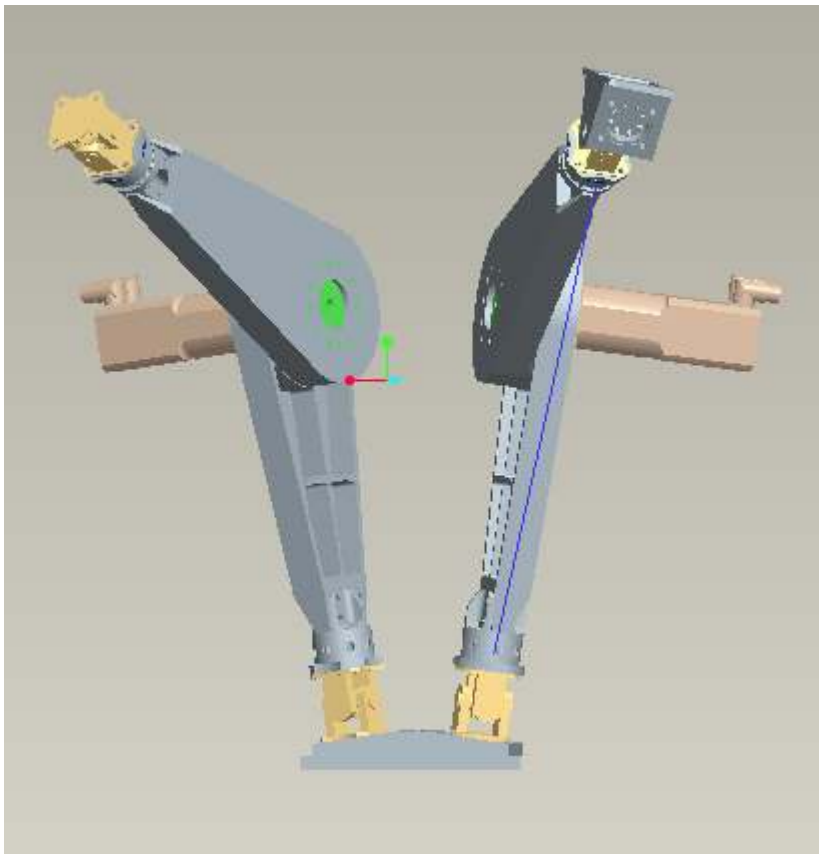

Figure 2.16 : Pro Engineer drawing of a leg pair.

3. SIMULATION OF THE HEXAPOD PLATFORM

In this chapter, real life system situations are modeled on a computer so that it can be studied to see how the system works.

Simulation of a system supplies practical information while designing real life systems. It allows the designer to determine the correctness and efficiency of a design before the system is practically assembled. Analyzing the effects of a design action in design period makes the system cost much cheaper than assembly period. Also, simulation allows the designer to examine a trouble at different stages of concept. By analyzing a system at a higher level of concept, the designer is better able to recognize the performance and the cooperation of all the high-level elements within the system. Therefore, the system is better equipped to remove the overall system complexity [9].

3.1 Modelling the System in MATLAB, SimMechanics

SimMechanics software is a block diagram-modelling environment, which uses the standard Newtonian dynamics of forces and torques for the design and simulation of rigid body mechanisms and their motions. The designer is able to model and simulate mechanisms with a set of tools to specify bodies and their mass properties, mechanisms' possible motions, kinematic constraints, and coordinate systems, and to initiate and measure body motions. A mechanical system can be represented by connected block diagrams, like other Simulink models. Hierarchical subsystems can be integrated. The visualization tools of SimMechanics software demonstrate and animate simplified renderings of 3-D machines by using the MATLAB Graphics system during the simulation [10].

SimMechanics software gives several ways to simulate and analyze mechanism motion in the Simulink environment. It provides a modelling environment for building three-dimensional rigid-body mechanical systems, includes a diversity of simulation techniques to analyze motion and size mechanical components, enables the implementation of high fidelity, nonlinear plant models in Simulink to sustain the

development and testing of control systems and supports Solidworks and Pro/ENGINEER CAD in order to specify mechanical models [11].

Linear plant models can be extracted from nonlinear plant model using Simulink Control Design tool. SimMechanics supports forward dynamics, inverse dynamics and kinematics, trimming and linearization motion analysis modes that allow the designer to test mechanism performance, develop appropriate controllers and select right actuation systems. Forward dynamics assigns forces, torques to the actuators, and calculates the consequential motions of the whole system. Inverse dynamics and kinematics decides the forces and torques that must be applied by the actuators to produce desired motion trajectories. Trimming identifies the steady-state equilibrium points to be used for linearization. Linearization extracts a linear model that estimates the system's response in terms of driving forces, joint and constraint configurations, and initial conditions [12].

The system simulation model is developed by the help of Smith and Wendlandt's Stewart Platform model [13]. The simulation model of the system is shown in Figure 3.1.

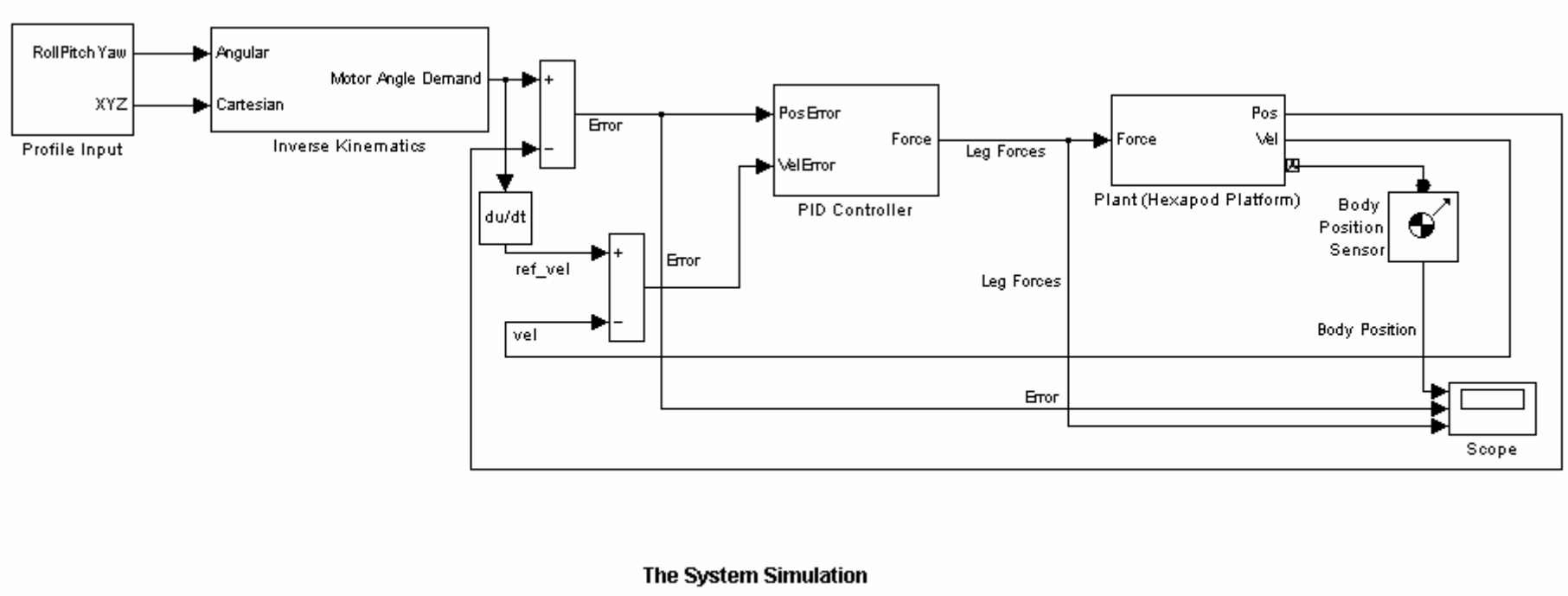

Figure 3.1 : The simulation model of the system.

The reference trajectory for the mobile platform is generated by the profile input subsystem as shown in Figure 3.2. In this model block, translational and rotational profile signals are created by using constant and sinusoidal wave Simulink blocks. However, the reference trajectory can be generated for requirements of any profile by using other types of Simulink blocks like ramp, step, signal generator or user-defined functions. The system's top base has initial offset in z-axis (z_offset) due to start-up configuration of the Hexapod Platform. All the reference signals are multiplied by gain to be able to turn on/off each component individually. The profile input subsystem has two outputs, namely, XYZ (translational) and RollPitchYaw (rotational) 3x1 matrices.

The next block is Inverse Kinematics for which details are given in Section 2.2.1. After the inverse kinematics process, motor angle demands for each leg are obtained.

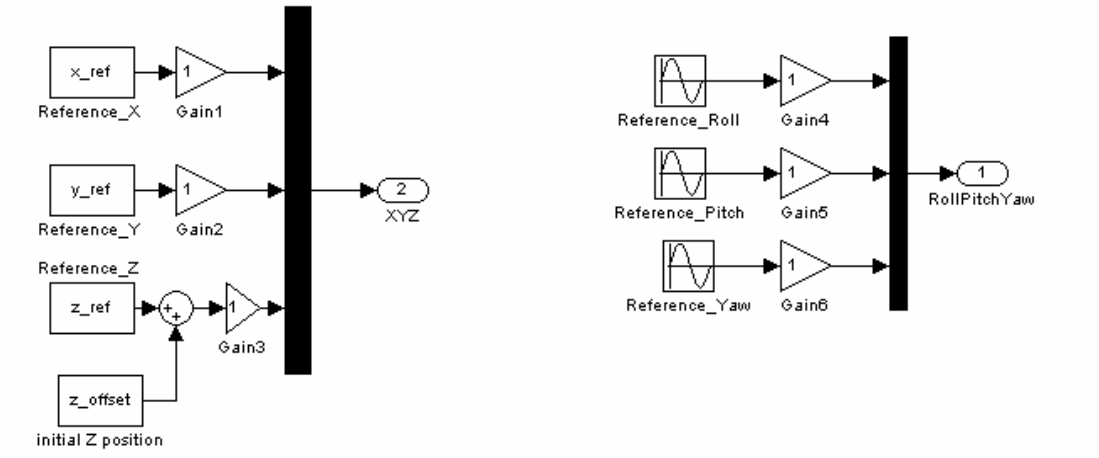

Figure 3.2 : The profile input subsystem.

The calculation of the motion errors ($\theta_{error\,i}$) for each revolute joint can be written as the following equation which is the difference of the reference angle of the leg and its actual or measured length.

$$\theta_{error\,i} = \theta_{reference\_joint\_angle_i} - \theta_{measured\_joint\_angle_i} \quad \textbf{(3.1)}$$

A PID controller is used to have the mobile platform follow the reference trajectory. A joint sensor measures the angular position used for calculation of motion error and angular velocity. The proportional, integral, and derivative terms repress the motion errors and get the mobile platform to follow the reference trajectory. Measured angular velocity is used to generate control signal instead of derivation block due to

aggressive response of the derivation process. Applied force to the actuator F_i is the output of the PID controller block. If $\theta_{error\,i}$ is negative, $\theta_{measured\_joint\_angle_i}$ is wide, and F_i is negative (leg length is shortened). If $\theta_{error\,i}$ is positive, $\theta_{measured\_joint\_angle_i}$ is narrow, and F_i is positive (leg length is lengthened). If $\theta_{error\,i}$ is zero, $\theta_{measured\_joint\_angle_i}$ equals the reference joint angle $\theta_{reference\_joint\_angle_i}$ and F_i is zero (leg length does not change). The PID controller is common for each leg because of identical configurations of the six legs. Different controller algorithms can be applied for each one if it is necessary. In Section 3.2, controller design is discussed in detail.

$$F_i = \theta_{error\,i} \times K_p + \int (\theta_{error\,i} dt) \times K_i + \dot{\theta}_{measured\_joint\_angular\_velocity_i} \times K_d \qquad \textbf{(3.2)}$$

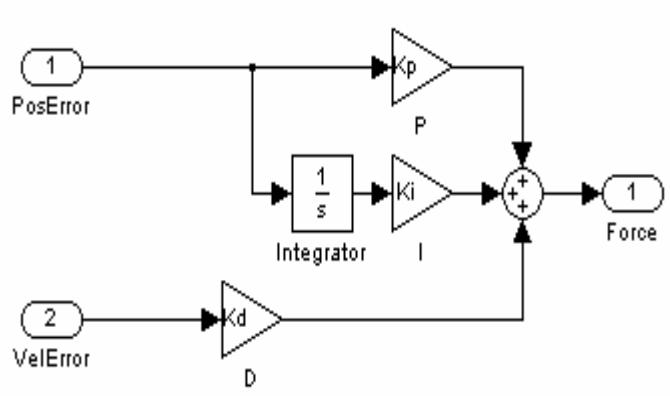

Figure 3.3 : The PID controller subsystem.

The next block is plant model for which details are given in Section 2.3.1. After the plant model block, the revolute joint angular positions and angular velocities measured by joint sensor are fed back to the controller. Finally, the overall system simulation model is ready to analyze system performance and improve the design.

3.2 Designing Controller for the Physical Plant

One of the most important problems of a system is the controller design. The aim of the controller design is to minimize the difference between the desired profile input and the system motion output. A feedback control law can be performed to drive the Hexapod Platform to a directed position. The implemented control form is (SISO) single-in, single-out. Symmetrically, defining controller parameters with one leg's

control behaviour is acceptable for the other five identical legs. The SISO approach ignores coupling between legs.

First of all, the physical plant modelling for a leg has to be practiced. Danaher synchronous servo motor (DBL3N00300 type) is used as actuator that is driven by a servo amplifier from SERVOSTAR 600 series under analog-speed control mode. Plant modelling is achieved by using Agilent 35670A FFT Dynamic Signal Analyzer.

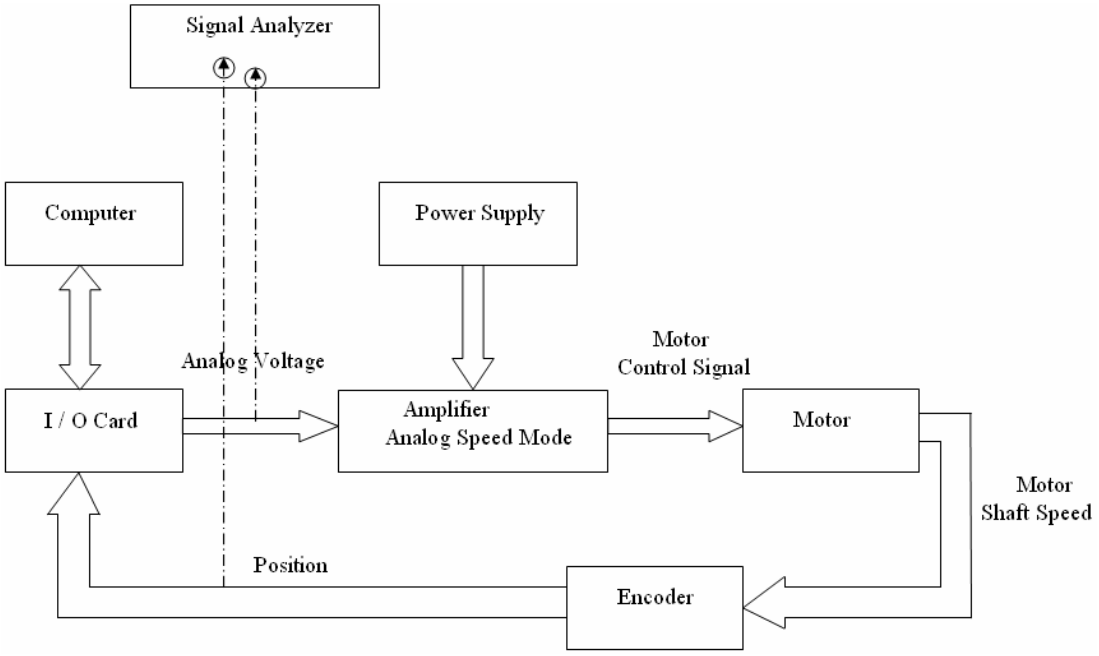

Figure 3.4 : Physical plant modelling of a leg.

$$G(s) = \frac{\theta(s)}{V(s)} = \frac{329.511}{s(s+1.124)} \qquad \textbf{(3.3)}$$

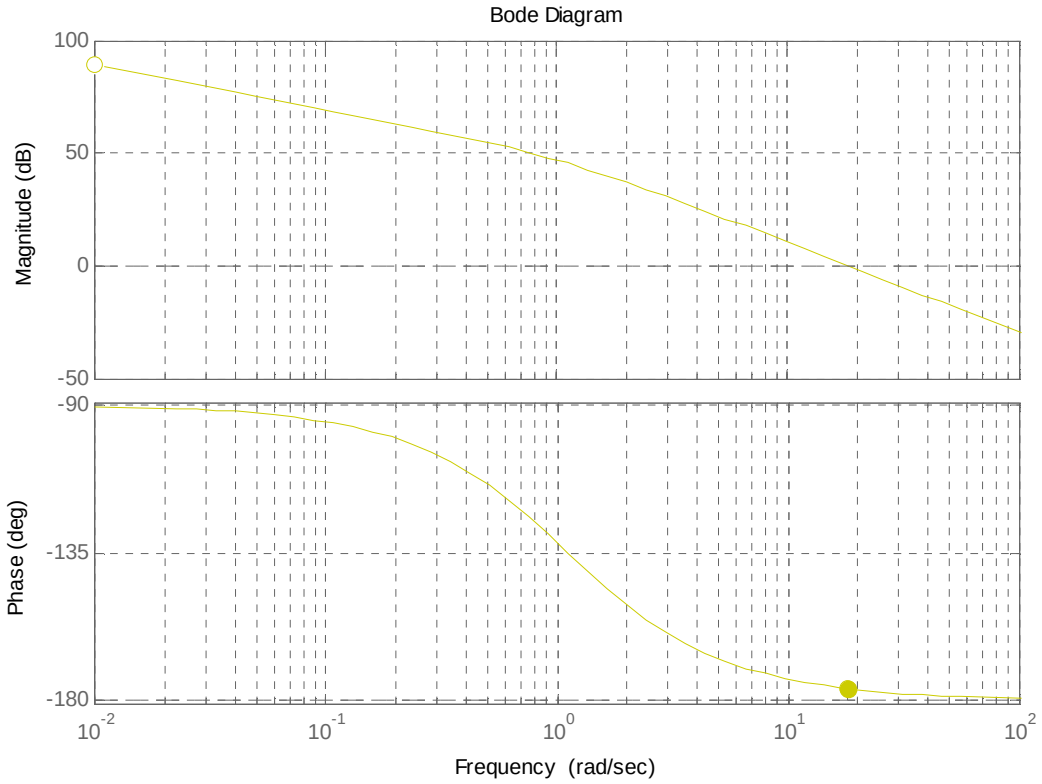

Figure 3.5 : Open loop bode diagram of G(s).

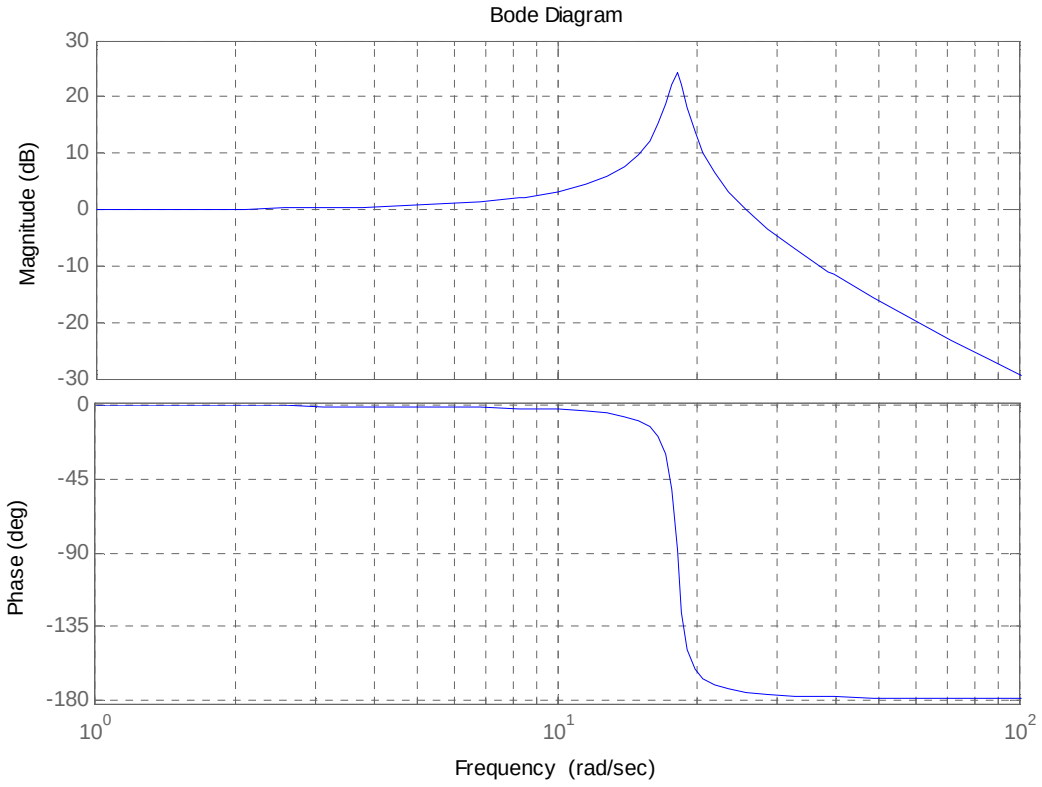

Figure 3.6 : Closed loop bode diagram of G(s).

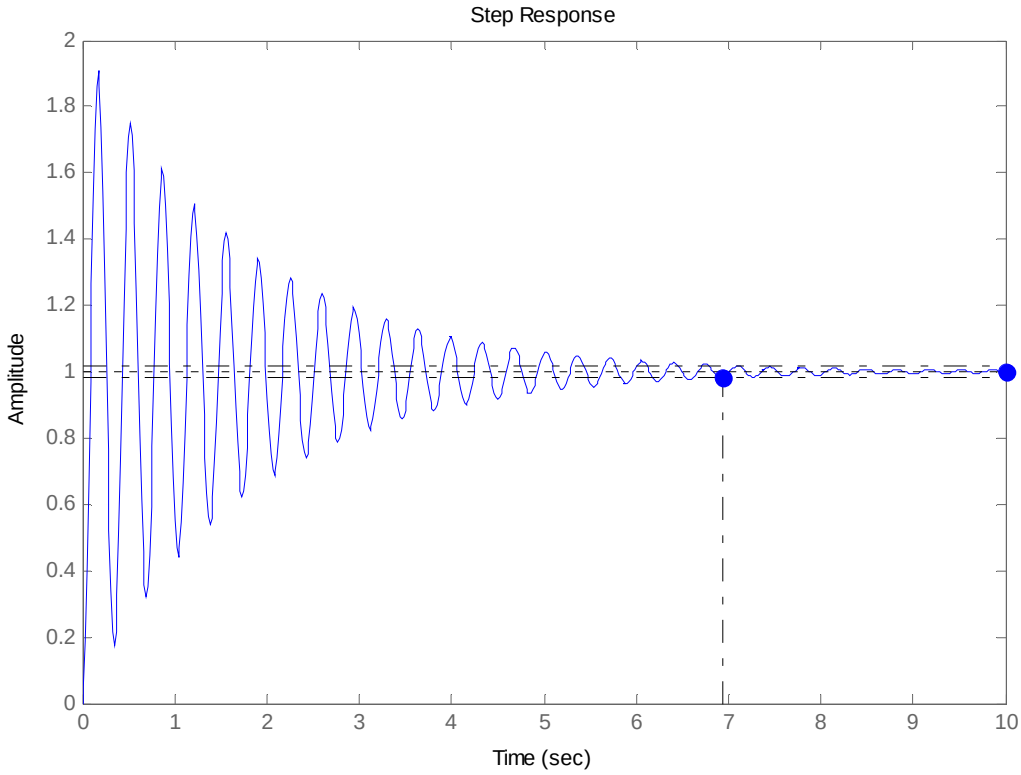

Figure 3.7 : Step response of G(s).

Open loop poles of the G(s) locate at 0 and -1.124. However, uncompensated system closed loop poles locate at -0.56 + 18.1i and -0.56 - 18.1i.

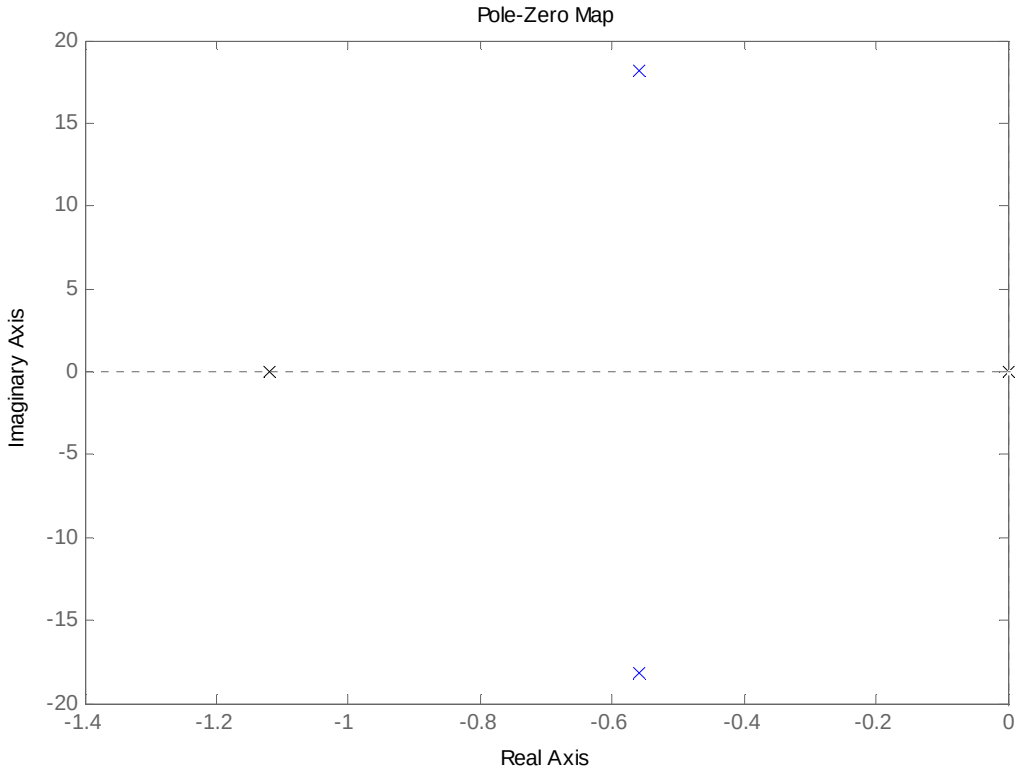

Figure 3.8 : Open loop (black), closed loop (blue) pole map.

The poles' positions of the physical plant model show that the system is stable, both poles lie on the left half side of the s-plane as shown in Figure 3.8. The first step in designing controller for any plant is observing the closed loop unity feedback response to check for stability. Many systems are unstable in open loop but stable in the closed loop.

The poles play a very critical role not only in the stability, but also in the dynamic response, of linear systems. By knowing poles of a system, we can have a rough idea about the response of the system to a given input. This means that in controlling a linear time-invariant system if we can determine the poles of the system under closed loop conditions, we will have a firm idea about the stability and dynamic response of the closed loop system. In fact, it has been shown that under certain conditions it is possible to assign the closed loop poles arbitrarily. The process of determining the closed loop system poles is therefore conveniently called pole assignment [14].

Reshaping of the system root locus by using pole placement technique is able to make the system behaved according to design specifications.

The desired response for the system has following characteristics. The settling time must be less than 0.3 second and the overshoot must be less than 20%, which implies that the damping ratio must be greater than 0.5 for the transient response of the system. Reference input of the system includes sinusoidal components, so the input can be thought as a ramp signal. The steady state error for the ramp signal must be zero.

A PID controller can be designed by using pole placement technique so that the response of the closed loop system meets the upper specifications. The general third order desired polynomial can be written in the following form.

$$Pds = (s^2 + 2\xi w_n s + w_n^2)(s + e) \quad \textbf{(3.4)}$$

$$w_n = \frac{4}{\xi * T_s} \quad \textbf{(3.5)}$$

$$\begin{aligned} 0 &= s^2 + 2\xi w_n s + w_n^2 = \mathrm{s}^2 + 32\,\mathrm{s} + 256 \\ 0 &= (s - (-16))(s - (-16)) \end{aligned} \quad \textbf{(3.6)}$$

Here ξ equals to 1 and settling time T_s equals to 0.25 that means that w_n equals to 16. The closed loop characteristic polynomial of the system is third order so the desired closed loop polynomial should be third order. The desired characteristic equation has two roots at -16. Also, an additional root has to be assigned to equalize the orders. The additional root should put at a distance five times from desired roots. Therefore, the desired closed loop polynomial, Pds, can be obtained as the following equation.

$$\begin{aligned} Pds &= (s - (-16))(s - (-16))(s - (-80) \\ Pds &= \mathrm{s}^3 + 112\,\mathrm{s}^2 + 2816\,\mathrm{s} + 20480 \end{aligned} \quad \textbf{(3.7)}$$

The next step is to find the PID controller coefficients by equalizing the closed loop characteristic polynomial of the system and the desired closed loop polynomial. The transfer function of the PID controller can be represented as follows.

$$PID = \frac{Kd\,\mathrm{s}^2 + \mathrm{Kp}\,\mathrm{s} + \mathrm{Ki}}{s} \quad \textbf{(3.8)}$$

poleassign.m, a function based on pole placement technique written in MATLAB, and its usage are given in Appendix C.2. Kd, Kp and Ki coefficients can be obtained

as 0.3365, 8.546 and 62.16, respectively by typing the following commands in the MATLAB command window.

```
syms  Kd Kp Ki;

C_Num = [Kd Kp Ki];

C_Denom = [1 0];

P_Num = 329.511;

P_Denom = [1 1.12 0];

CL_Poles = [-16 -16 -80];

TransferFunctionOfPID=poleassign(P_Num, P_Denom, CL_Poles, C_Num,
C_Denom);
```

The following equation shows the transfer function of the PID controller.

$$PID = \frac{0.3365\,s^2 + 8.646\,s + 62.16}{s} \qquad \textbf{(3.9)}$$

The system root locus is reshaped by the PID controller.

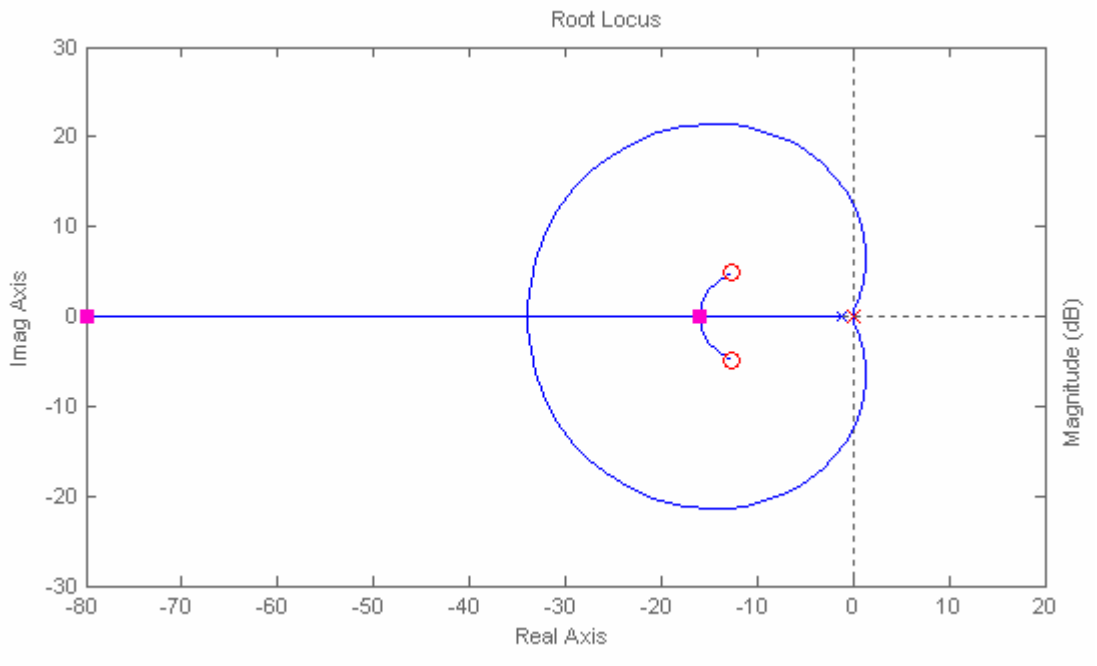

Figure 3.9 : Reshaped root locus of the system.

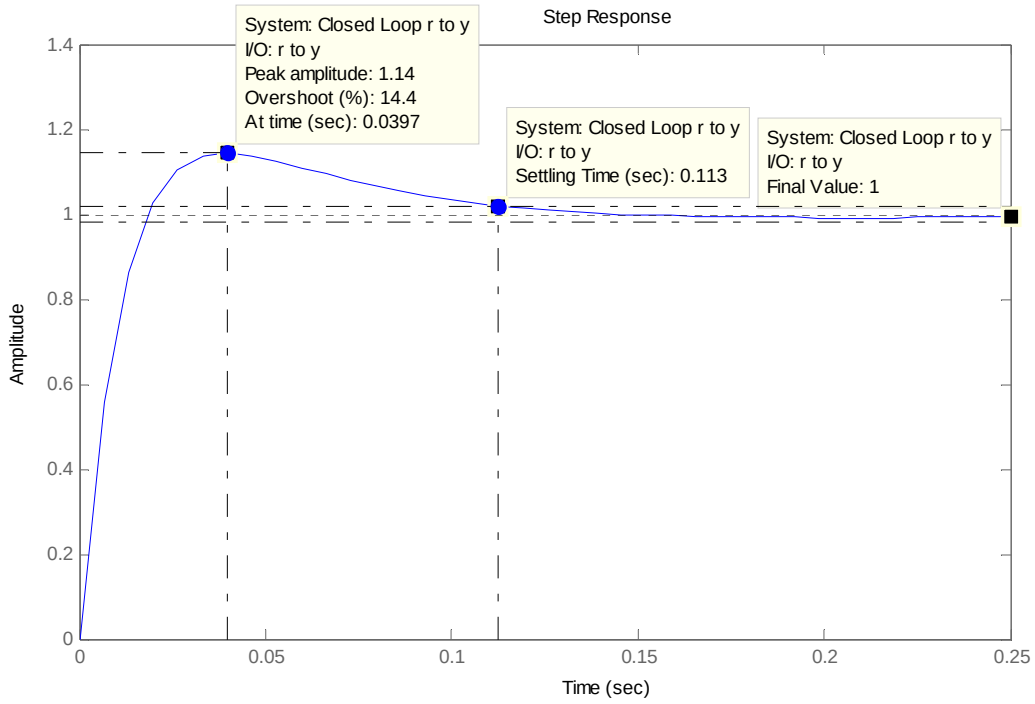

Figure 3.10 : Step response of the system with controller.

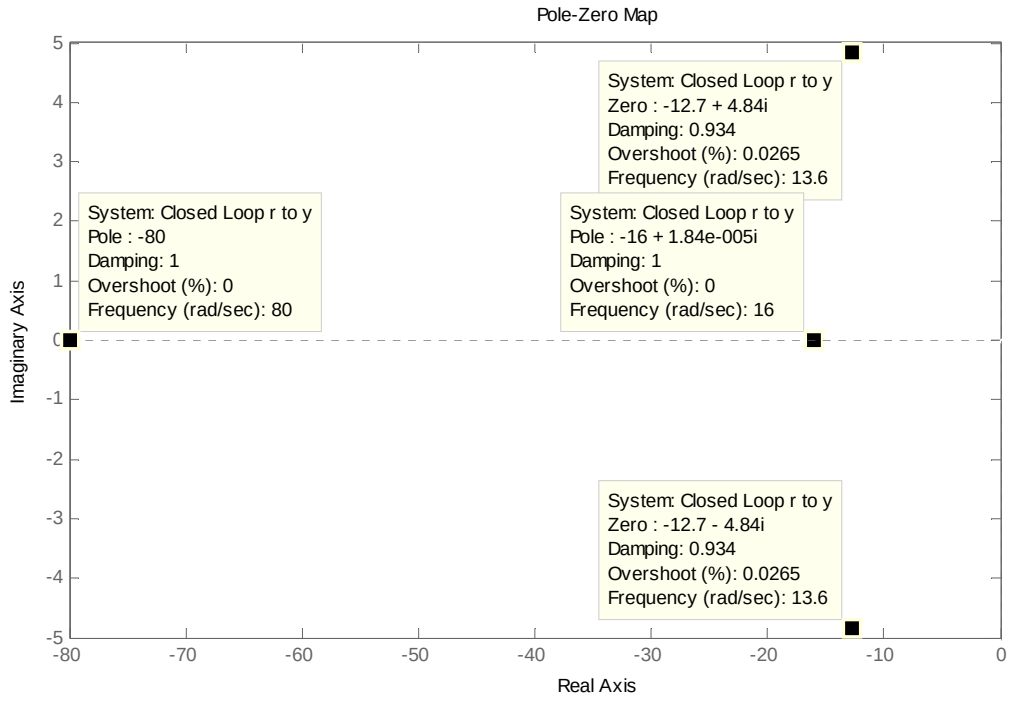

Figure 3.11 : Closed loop pole-zero map.

Closed loop poles are located at -80, -16 and -16 as mentioned in design specifications. New settling time of the system is 0.113 second and overshoot is

14.4%. Also, steady state error for ramp signal is zero because the system type is two as seen the following equation.

$$G_{Controller} * G_{System} = G_C(s)G_S(s) = \frac{110.9\,s^2 + 2816\,s + 20480}{s^2(s+1.12)} \quad \textbf{(3.10)}$$

Ramp function in Laplace domain can be written as equation (3.11).

$$Ramp(s) = \frac{R}{s^2} \quad \textbf{(3.11)}$$

$$e_{SS}(s) = \lim_{S\to 0} sE(s) = \lim_{S\to 0} s\frac{R(s)}{1+G_C(s)G_S(s)} = \frac{R}{\lim_{S\to 0} sG_C(s)G_S(s)} \quad \textbf{(3.12)}$$

$\lim_{S\to 0} sG_C(s)G_S(s)$ equals to infinite, so e_{SS} equals to 0, which means there is no steady state error for ramp signal reference.

The controller is implemented on the physical plant. The settling time of the physical plant is 0.16 second and overshoot is 8.9% that satisfy the design specifications.

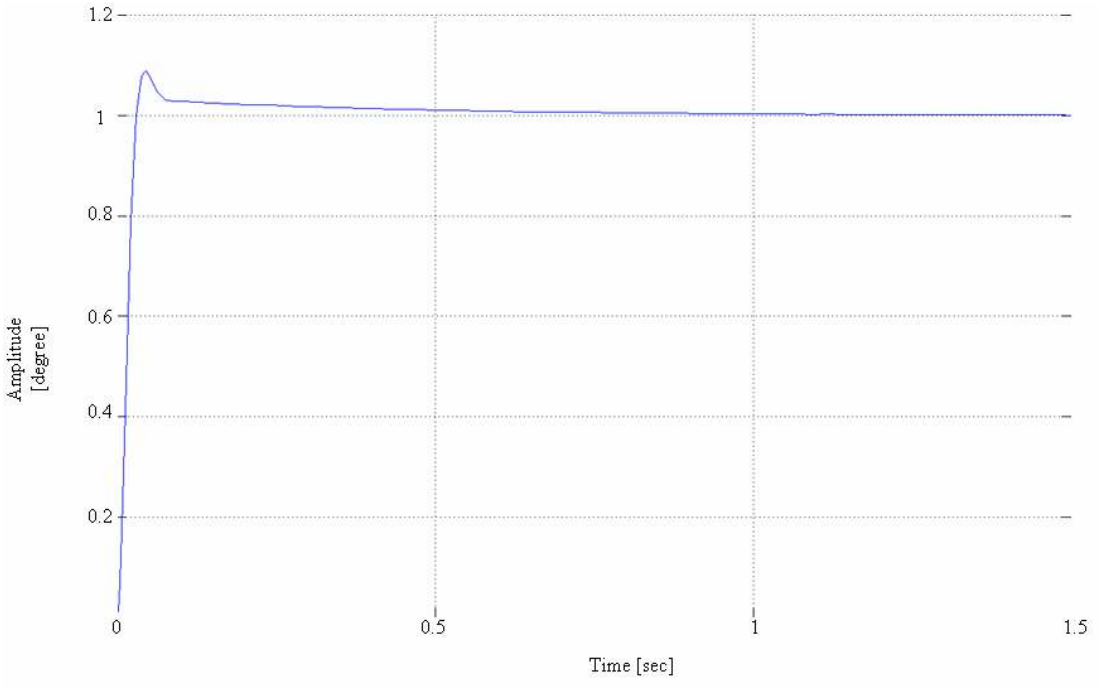

Figure 3.12 : Step response of the physical plant.

4. SYSTEM IMPLEMENTATION

The major purpose of this study is design a motion simulator platform to test stabilized head mirror of a tank. Used hardware and software facilitate embedded control system design. Embedded real time control is achieved by using MATLAB, Simulink, xPC Target Embedded Option and compatible devices. xPC Target [15] turns personal computer hardware into a rapid prototyping platform.

The platform can be used to build, for example, an aircraft simulator or a tank simulator. The legs are used to move the simulated tank so as to give the impression of being inside an actual tank. The platform is called as “hexapod,” due to its six legs.

Figure 4.1 : The motion simulator and SHM.

4.1 Characteristics of Embedded Control System

The following figure shows an embedded control system's common structure.

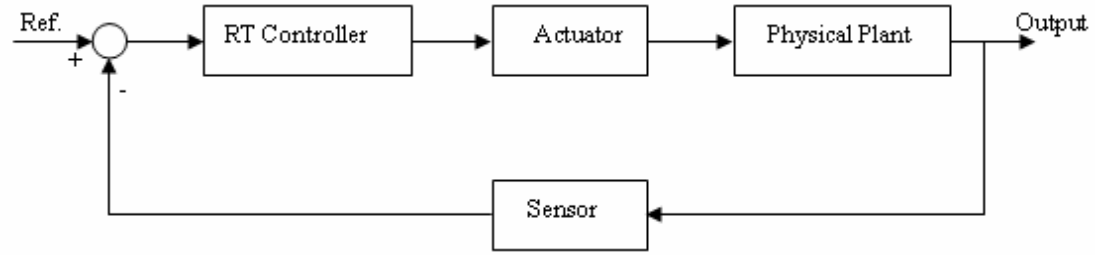

Figure 4.2 : Close loop system block diagram.

The physical plant operates in high power electricity domain such as mechanics, thermal, pneumatics and other domains. However, the controller operates in low power electricity domain. The converters are essential for transaction between high power domain and low power domain that the transducers and sensors take roles in this issue. For our hexapod platform setup, each leg is equipped with a servomotor that extends angle between the upper leg and lower leg. Low power microprocessor executes the control law that computes the desired position. An amplifier changes this electrical signal into a high power equivalent, which drives the motor. A sensor measures the angle between the upper leg and lower leg. The angle is calculated by the fact that the encoder sends pulse signals and hardware counter counts the number of the pulses. Also, the necessity of the transformation between continuous time and discrete time is obvious. The physical plant operates in continuous time; however, the controller has a clock whose value changes only at discrete points in time.

Rapid prototyping is a vital issue in scientific research and education. One of the most important benefits of rapid prototyping is a quick way to validate the controller code by executing it with the sensors, actuators and physical plant or plant model [16]. The following figure shows the rapid prototyping configuration of the Hexapod platform setup.

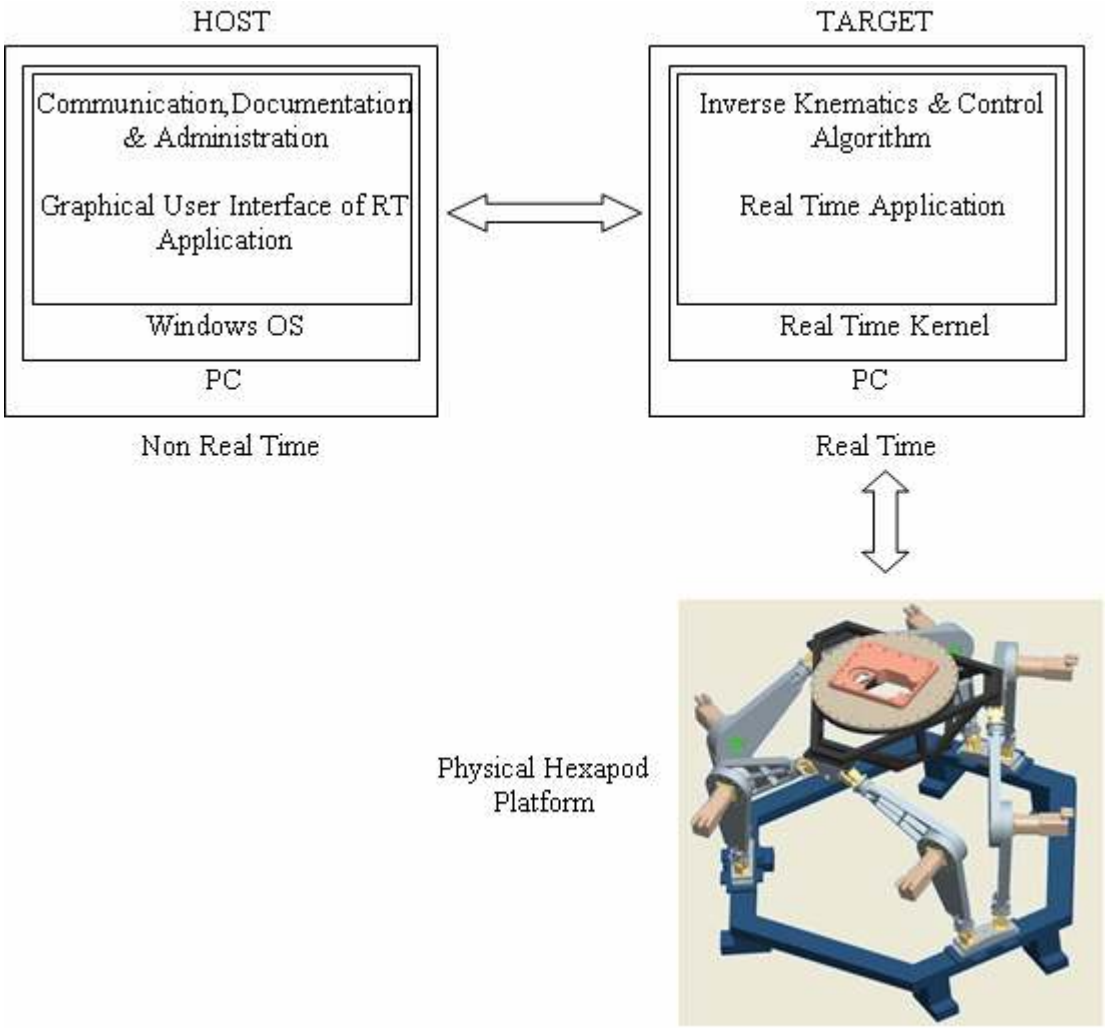

Figure 4.3 : Configuration of the motion simulation platform.

4.2 Hardware Implementation

xPC Target makes a general-purpose PC hardware into a prototyping environment that can be used for data acquisition, rapid prototyping and hardware in the loop (HIL) simulation [15]. Two computers exist in the configuration of the motion simulation platform. A rack-mounted non real time host PC is setup for GUI applications, communication with RT application of target PC and six servo amplifiers, system event logs and documentation. As the real time controller, an industrial PC with integrated monitor is used for processing control algorithm, inverse kinematics calculation and driving servo amplifiers in analog-speed mode. PC-based platforms present flexibility, expandability and scalable computing power for the designers. xPC Target can run on any computer for real time applications.

Real Time PC configuration includes 2.4 GHz Intel Pentium IV floating-point processor and 512 MB Ram.

4.2.1 Actuators

Actuating mechanism is achieved by motors, which are mounted on each leg of the Hexapod Platform. These six motors are chosen from Danaher synchronous servo motor DBL3N00300 series. The motors are driven by servo amplifiers from SERVOSTAR 600 series, which receive the control voltages from the analog output channels of Humusoft MF 624 multifunction I / O card. The amplifiers' input is low power analog signal, which is converted into high power sinusoidal voltage at the output of the amplifiers to drive the motors. Appendix A.1 describes the specifications of the motor and the amplifier.

The Hexapod Platform has six legs, so six amplifiers and six motors are used to orientate the platform to the desired position. The platform is a multi-axis system that synchronization and communication is so important. A computer commands the master amplifier over RS232 protocol and other five amplifiers over CAN protocol. The master amplifier receives the motion commands, then transfers to the others over RT CAN bus and all drivers execute the commands simultaneously. Figure 4.5 and Figure 4.6 describe the multi-axis system.

Motion control is a big part of the implementation of the designed system. Servo Star 600 is easy to specify, install and operate. The system performance easily meets the design specifications with the amplifier, which lets the designer much concentrate on design solution instead of struggling with motor driving configuration. Figure 4.4 shows the details of SERVOSTAR 600 amplifier.

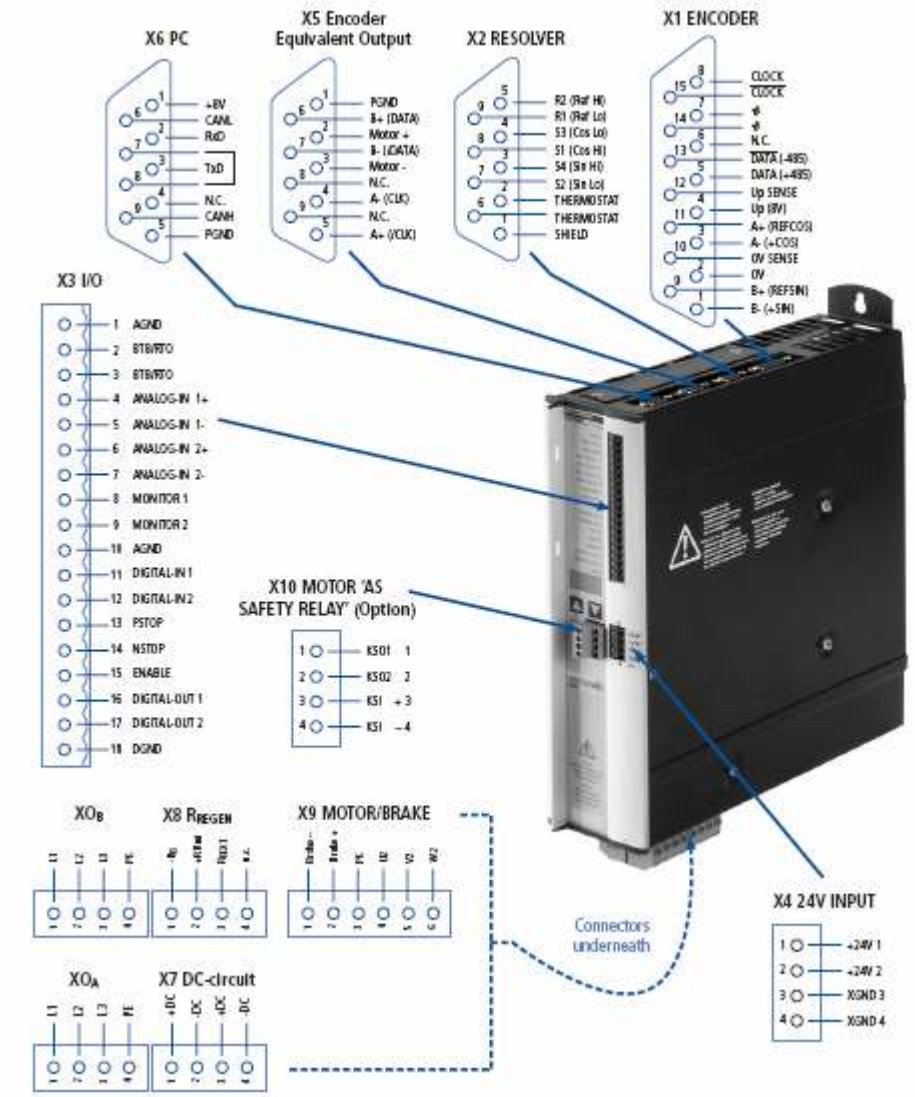

Figure 4.4 : Servo-Star 600 motor driver.

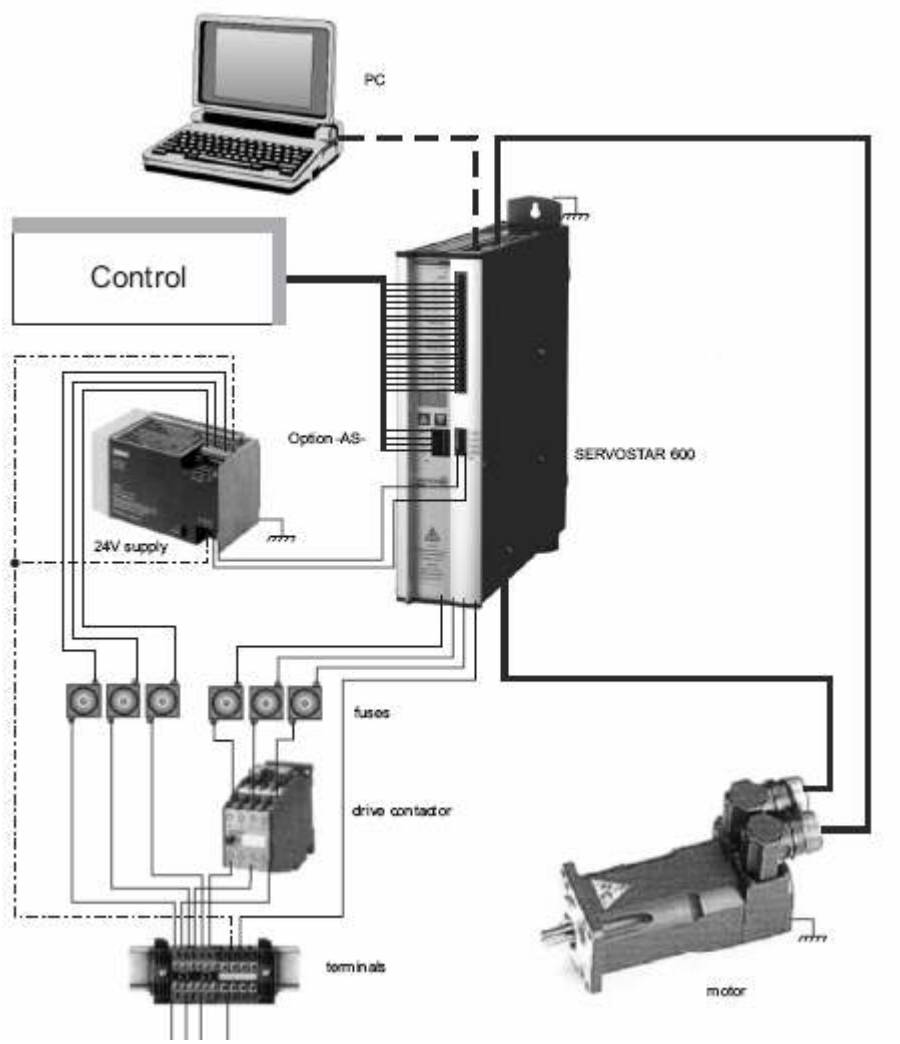

Figure 4.5 : Components of a servo system.

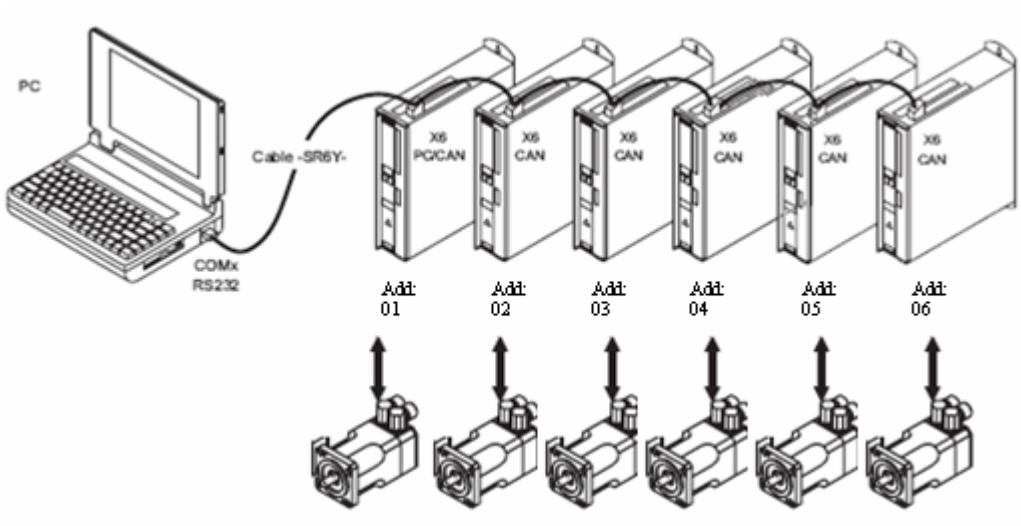

Figure 4.6 : Multi-axis system.

4.2.2 Reductors

Reductor is a dynamic conveying mechanism, which using gear speed converter to reduce winding number of electric engine (motor) to desired level and gain bigger torque. At present reductor is widely used in dynamic force and movement conveying mechanism. Its traces could be find in all type of mechanical driving system, from vehicles as ship, car and locomotive, constructional heavy machines, processing machines in mechanical industries and auto production equipments, to normal household appliances, clocks and so on. Its application covers from giant power transmission to small load and accurate angle transmission. In addition, reductor can decrease speed and increase torque when used in industry, it is widely used in speed and torque transferring machine.

Used reductor with 1:89 gear ratio on the leg of the Hexapod Platform increases applied torque 89 times and decreases taken position and speed 89 times. This kind of mechanical transmission can cause backlash problem during the implementation. Backlash problem in the system invites noise, precision loss in positioning, uncertainty and delay. Backlash makes the position loop less stable and prone to oscillation. Backlash is an undesirable characteristic and should be minimized by proper maintenance [17].

4.2.3 Resolvers

A resolver is built into the motors as standard feedback element. The servo amplifiers in the SERVOSTAR series evaluate the resolver position and supply sinusoidal currents to the motors [18]. The resolver uses an optical beam, which produces a sequence of electrical pulses to measure the rotation.

The resolver should not be reset at an arbitrary position, because the pulse counting starts when the system is powered up. The solution is to drive the motors to the upper security switches of the actuators, then jog to home position (predefined steps from the upper security switch) and define that position as zero location. The system is turned on and off at home position to prevent miscalculation of the motor angles. A reference marker shown in Figure 4.12 calibrates the zero location.

The resolver produces the electrical pulses with a power and impedance that permit it to be directly connected to a counter board. The real time computer with xPC target embedded option has two Humusoft MF 624 I / O cards. A single Humusoft MF 624

card has four quadrature encoder inputs with single-ended or differential interface. Inputs are differential TTL compatible with Schmitt triggers [19]. The encoder inputs read 16384 counts per a full turn in the implementation. The precision is 0.02197265625 degree per count. Mounting a reductor on an actuator provides higher precision value. The precision with 1:89 gear ratio becomes 0.000246883 degrees per count. Wiring details can be found in Appendix A.2.

4.2.4 I/O cards

I/O device takes an important role to turn a simple PC into a high capable digital controller. Humusoft MF 624 card is chosen specifically as data acquisition device since it is compatible with MATLAB xPC toolbox. MATLAB provides an interface for physical I/O devices and enables hardware in the loop applications. Humusoft MF 624 multifunction I/O card is designed for the need of connecting PC compatible computers to real world signals. The MF 624 contains 8 channel fast 14 bit A/D converter with simultaneous sample/hold circuit, 8 independent 14 bit D/A converters, 8 bit digital input port and 8 bit digital output port, 4 quadrature encoder inputs with single-ended or differential interface and 5 timers/counters. The card is designed for standard data acquisition and control applications and optimized for use with Real Time Toolbox for Simulink®. MF 624 features fully 32-bit architecture for fast throughput [19]. Hardware specifications can be found in Appendix A.3.

Figure 4.7 : MF 624 PCI card.

Our hardware configuration includes two MF 624 PCI cards which provide the following features.

- Six encoder inputs to the read motor angles;
- Six analog outputs to drive the servo motors in analog-speed mode;
- Four digital inputs for “RemoteStop” and “HardFailure” discrete-time signals (these two signals stop HIL simulation if stop button is pressed or a hardware failure such as amplifier fault occurs), and turning on/off (GUI application sends on/off discrete-time signal over the parallel port of the host PC to the digital input of MF624 card on the target PC) pitch and yaw reference signals (test profiles);
- Two digital outputs for HIL simulation running discrete-time signal and “ErrorStop” discrete-time signal (if the position on a leg is greater than 20 degrees, this signal stops HIL simulation and sends application stopped information over the digital output of MF624 card on target PC to the parallel port of the host PC).

4.2.5 Communication protocols

The host (non real time) pc has a serial and a parallel port. These ports can be used for commanding and data sending to the peripherals. The host pc communicates with the amplifiers over RS-232 protocols. RS-232 (RS stands for Recommended Standard) is widely used for asynchronous communication technique. Full description of the hardware requirements and software protocols is not given here. The interested reader is referred to Serial Port Complete book by Jan Axelson.

Data is transferred using the ASCII (American Standard Code for Information Interchange) standard between the host pc and the master amplifier. ASCII Object Reference SERVOSTAR 400/600 document describes protocol specifications and motor control commands such as position control mode, speed control mode, acceleration ramp, operational modes etc. Communication rate, which is called the baud rate (bits per second), is 9600 baud [21]. The pc controls the master amplifier over serial connection and the master controls the other five amplifiers over CAN connection.

CAN (Controller Area Network) is a serial communication protocol developed mainly for time critical industrial applications. The CAN protocol is optimized for short messages. Thus, the protocol is message oriented, and each message has a

specific priority that is used to arbitrate access to the bus in case of simultaneous transmission. The bit stream of a transmission is synchronized on the start bit, and the arbitration is performed on the following message identifier, in which a logic zero is dominant over a logic one. A node that wants to transmit a message waits until the bus is free and then starts to send the identifier of its message bit by bit. Conflicts for access to the bus are solved during transmission by an arbitration process at the bit level of the arbitration field, which is the initial part of each frame. Hence, if two devices want to send messages at the same time, they first continue to send the message frames and then listen to the network. If one of them receives a bit different from the one it sends out, it loses the right to continue to send its message, and the other wins the arbitration. With this method, an ongoing transmission is never corrupted. Figure 4.8 shows the message frame format of the CAN bus [22]. The interested reader is referred to Controller Area Network book by Konrad Etschberger for further information.

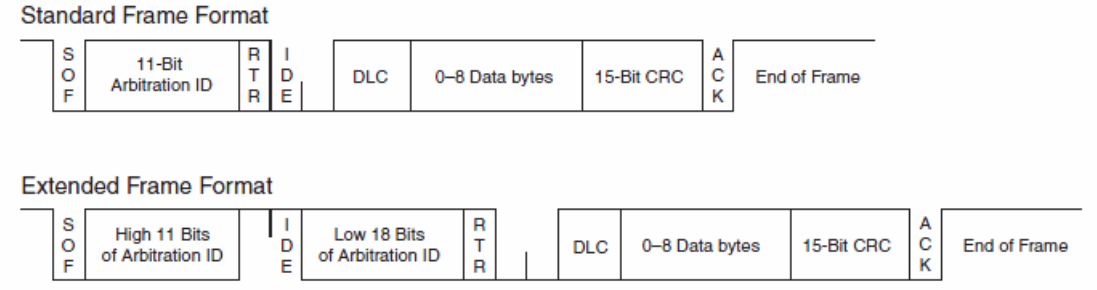

Figure 4.8 : Standard and extended CAN frame formats.

The host pc uses a parallel port to communicate with target RT application and to check unsafe situations, while the RT application is running, by writing and reading data to the parallel port. The parallel port has 8-bit data on corresponding pins of the port. All eight data is transferred in a cycle time. The following figure shows a parallel port pin diagram [23]. Full description of the hardware requirements and software protocols is not given here. The interested reader is referred to Parallel Port Complete book by Jan Axelson.

The main idea of the parallel port usage on the host PC is receiving digital signals from MF 624 card's digital outputs and transmitting digital signals to MF 624 card's digital inputs on the target RT PC. These six digital signals and their functions are describes in Section 4.2.4. Figure B.10 shows the connection between the parallel port and MF 624 DIO.

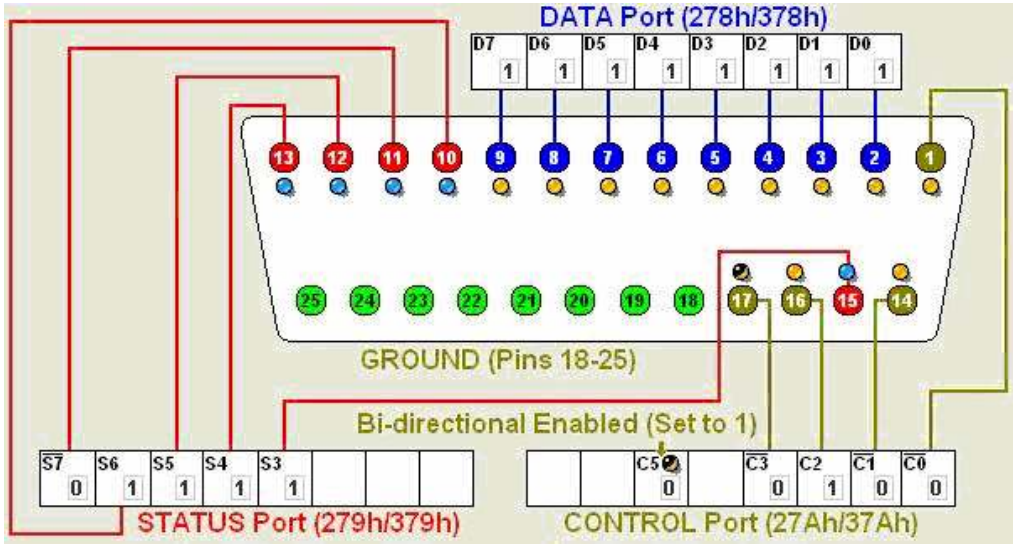

Figure 4.9 : Parallel port pin diagram.

General communication protocols and configuration can be expressed as the following figure.

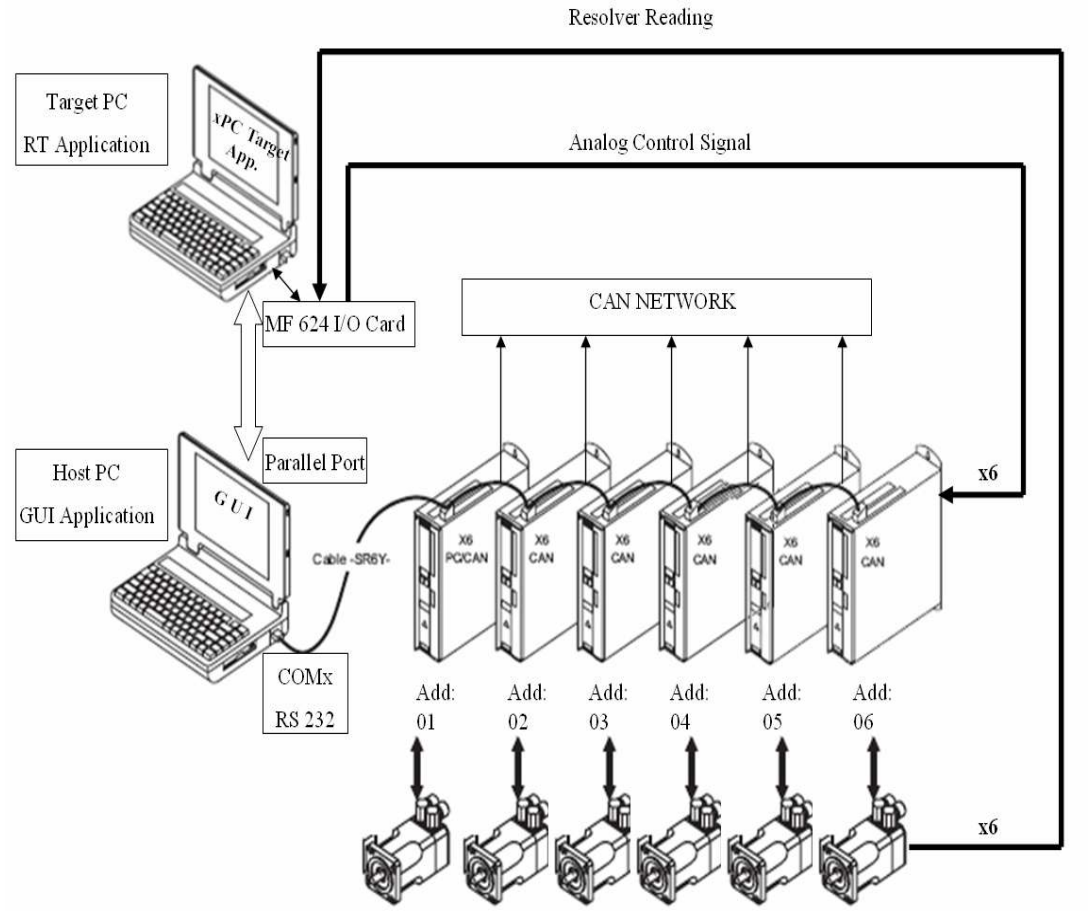

Figure 4.10 : Communication details.

4.3 Mechanical Design

Design details of the mechanical components are not issue of this study. ASELSAN does not permit to give specific information about design of mechanical components such as mass properties, actual volumes and geometrical shapes, inertias, materials and other component properties to avoid unauthorized reproduction of the Hexapod Platform.

4.4 Electrical Design

An electrical control panel is designed for emergency stop scenarios, indication of the platform status, manual manipulation and power distribution. The following figure shows the control panel of the Hexapod Platform.

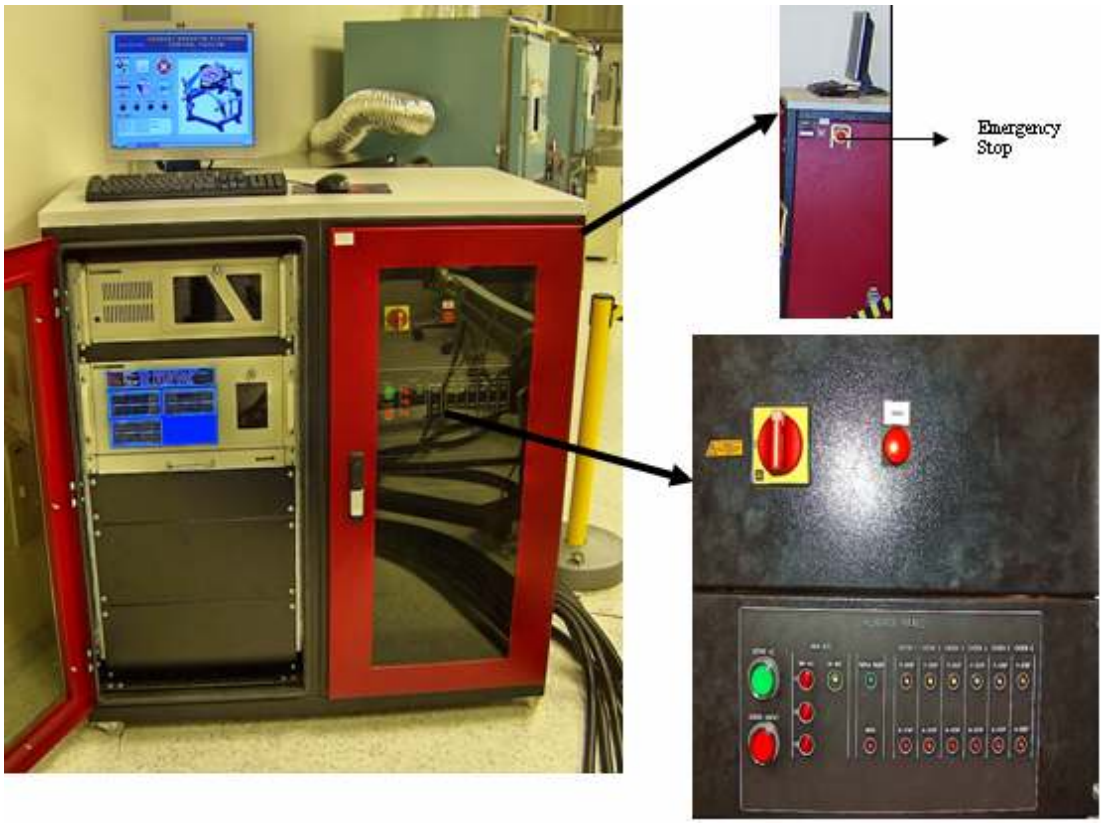

Figure 4.11 : Electrical control panel.

Schematic pages that include electrical design details are given in Appendix B.1. Maximum working angle is 60 degrees that defines the region between the upper and the lower security switches as shown in Figure 4.12. If the marker on any arm hits any of the security switches that are connected to servomotors' "limit-switch, PSTOP / NSTOP" inputs while the Hexapod Platform is simulating the tank motion, the

motors are locked at the legs' current positions by motor-holding brakes and the control panel gives alarm for the maintenance person. The power of the motors is disconnected when the displacement is higher than 20 degrees on any motor, a hardware failure on any servo amplifier occurs or emergency stop button is pressed for any reason.

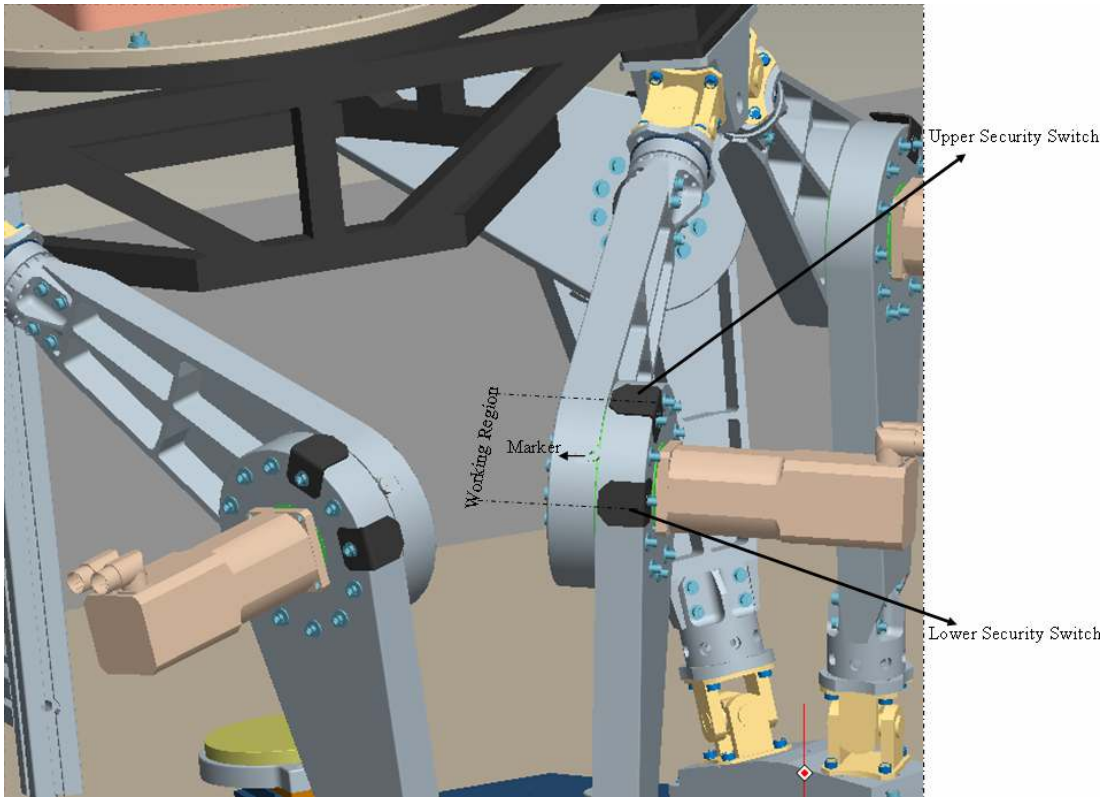

Figure 4.12 : Security switches on a leg.

4.5 Software Implementation

The Hexapod Platform and other systems are designed to test stabilization sensitivity of the stabilized head mirror of a tank. Reference profile of the motion simulator (Hexapod Platform) is constituted from motion of a tank on APG (Aberdeen Proving Ground). xPC target application on RT computer produces reference signals for top plate position in angular and Cartesian format. The inverse kinematics algorithm described in Section 2.2.1 in detail calculates motor angle demands for each leg of the Hexapod Platform. The application reads the encoder increments then actual motor angles are computed. The error signals for each leg are obtained by subtracting actual motor angles from motor angle demands. The PID controller block described in Section 3.2 in detail produces analog voltages for each motor of the motion

simulator according to error signals. The analog output block sends the voltages (corresponding motor speeds) to the amplifiers over MF 624 analog output. The error scopes block displays motor error angles on rack mounted RT computer. “System Digital Outs” block sends discrete signals to control panel and non-RT GUI computer. Finally, the stop block stops simulation if undesired events occur. The following figure shows the hardware in the loop model file that will be built and downloaded to the target RT computer.

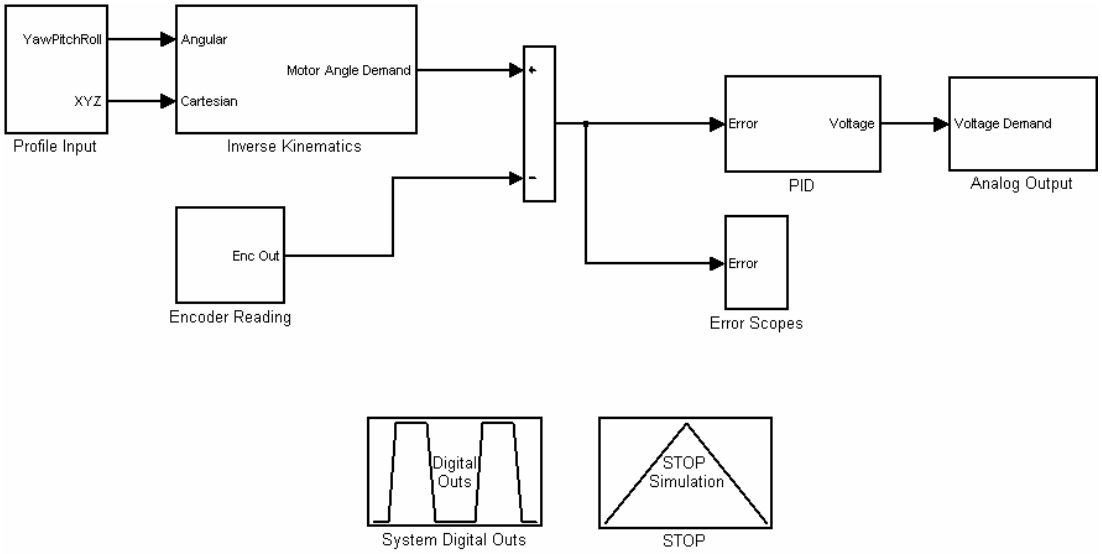

Figure 4.13 : HIL model for xPC target application.

Some constant definitions are needed such as leg and arm lengths and base and top platform coordinates variables, which are given in Appendix C.1.

4.5.1 Reference profile generation

The purpose of designing Hexapod Platform is to simulate a tank motion for the tests of stabilized head mirror.

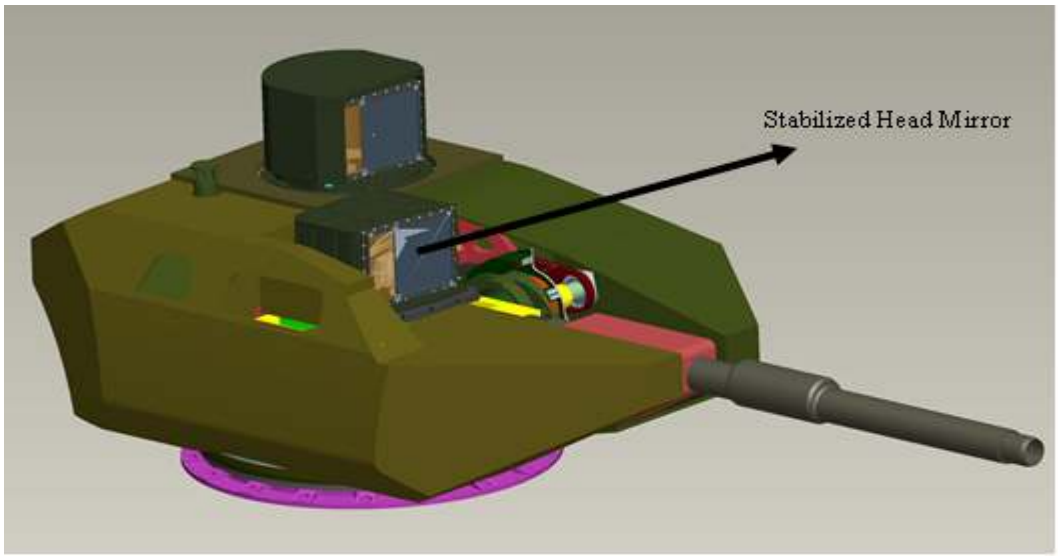

Figure 4.14 : SHM on a tank turret.

To accomplish motion simulation of a tank, the motion profile of the tank turret has to be defined. A track called Aberdeen Proving Ground (APG) is applied for the motion profile. Different barriers with different heights at specific positions are located on APG test track according to military standards.

The performance of the stabilized head mirror is mostly affected by the motion of the tank in pitch and yaw axis. The motion data of the tank is collected by sensors (gyroscopes) during the performance test on APG track. The velocity components in sinusoidal form are obtained by analyzing the collected motion data from gyro. The following tables describe the test profile of the stabilized head mirror in pitch and yaw axes, respectively.

Table 4.1 : Sinusoidal velocity components of pitch axis.

Component Number	1	2	3	4	5	6	7	8
Frequency (Hz)	1	2.5	5	10	17	23	30	40
Amplitude (rad/sec)	0.238	0.028	0.0084	0.0042	0.0123	0.045	0.021	0.0028

Table 4.2 : Sinusoidal velocity components of yaw axis.

Component Number	1	2	3	4
Frequency (Hz)	1	2.5	5	10
Amplitude (rad/sec)	0.238	0.028	0.0084	0.0042

The motion profile is created in MATLAB Simulink. The profile input model produces three rotational (Yaw, Pitch and Roll) and three translational positions. The distance from the center of the base platform to the center of the top platform of the Hexapod Platform is 0.66173 meters that is the offset of the z-axis position command.

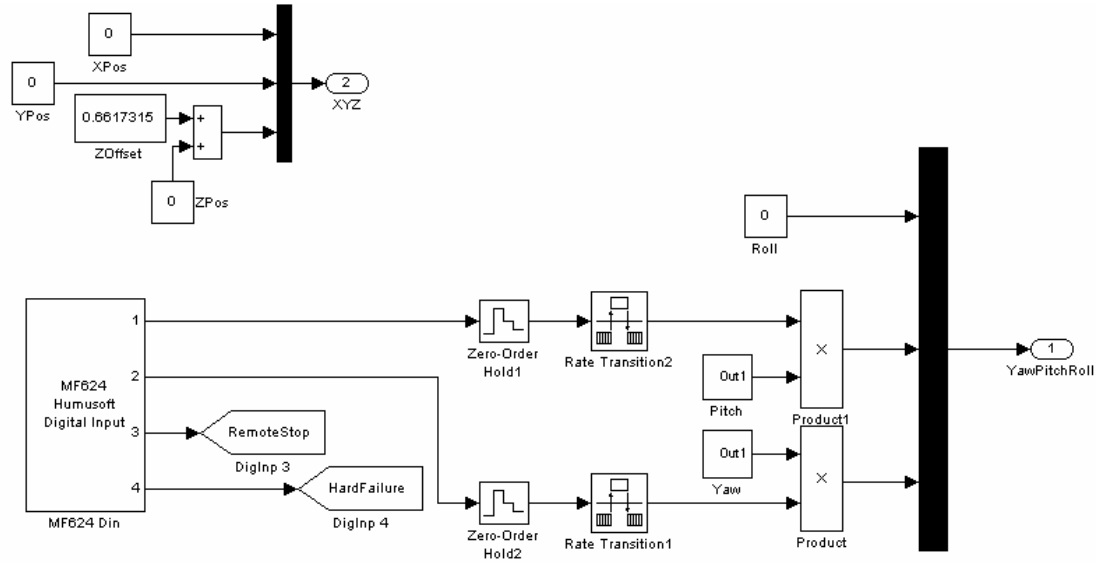

Figure 4.15 : Profile input subsystem.

There is no motion for x-axis, y-axis, z-axis and roll-axis. Pitch and yaw motions can be enabled individually by MF 624 digital input controlled by HBP_Kontrol software. Zero-order hold is placed to synchronize the starting point of the motion with zero crossing of the reference profile. Therefore, sampling time of the zero-order hold is chosen as one second that is the common time multiple of the sinusoidal components. This helps to avoid sudden jumps when the motion begins. Again, the motion is also stopped at home position .The following figures show the generation of pitch and yaw axes' reference positions by integration of the velocity data on the Table 4.1 and 4.2.

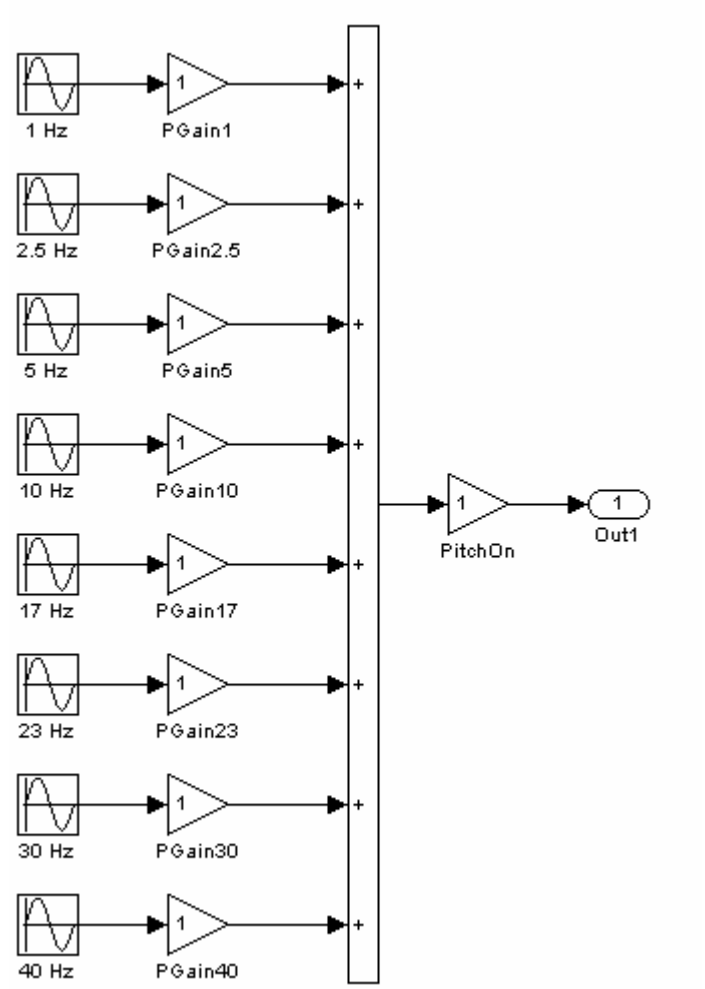

Figure 4.16 : Pitch motion.

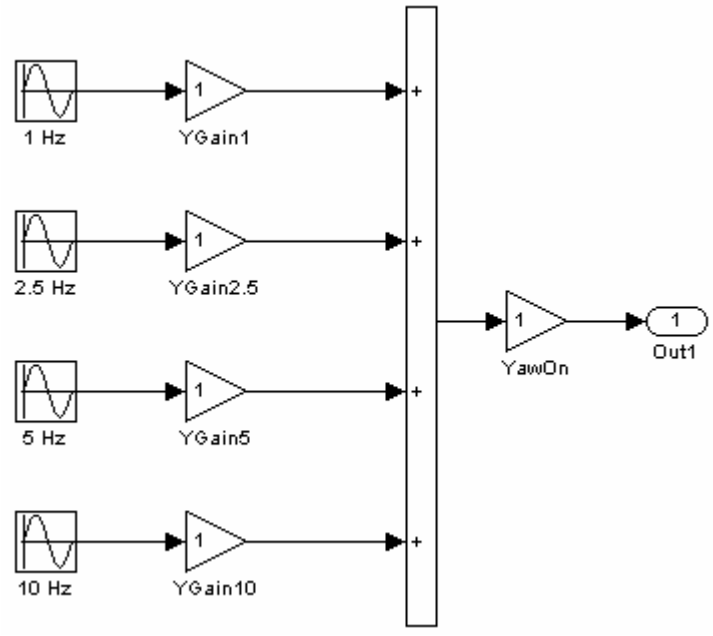

Figure 4.17 : Yaw motion.

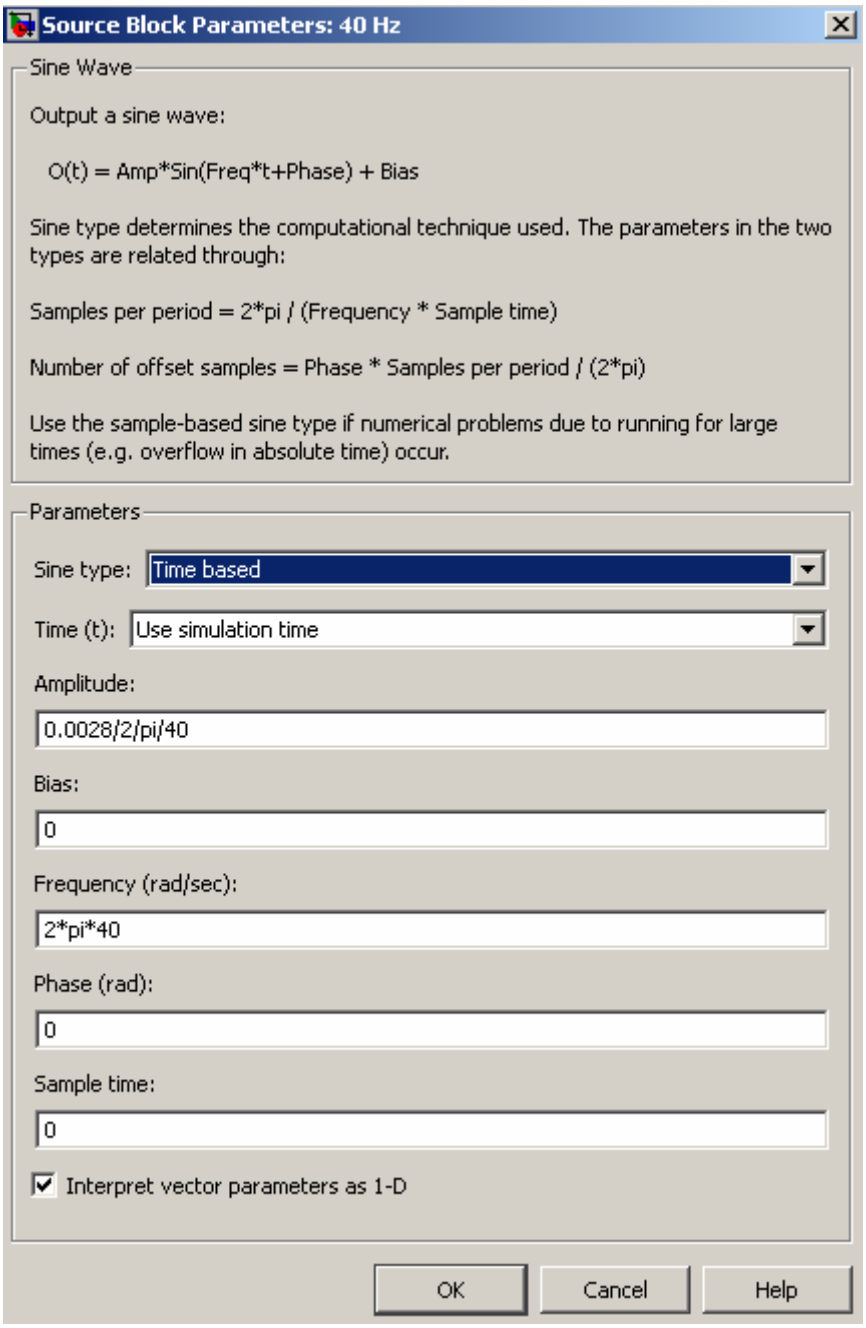

Figure 4.18 : 40 Hz pitch component block parameters.

4.5.2 Actual angle calculation of the legs

MF 624 encoder inputs count the resolver increments of the six legs. Motor resolver produces 16384 increments for a full turn (360 degrees). To calculate the leg angles, the reduction gear factor (1/89) also has to be included.

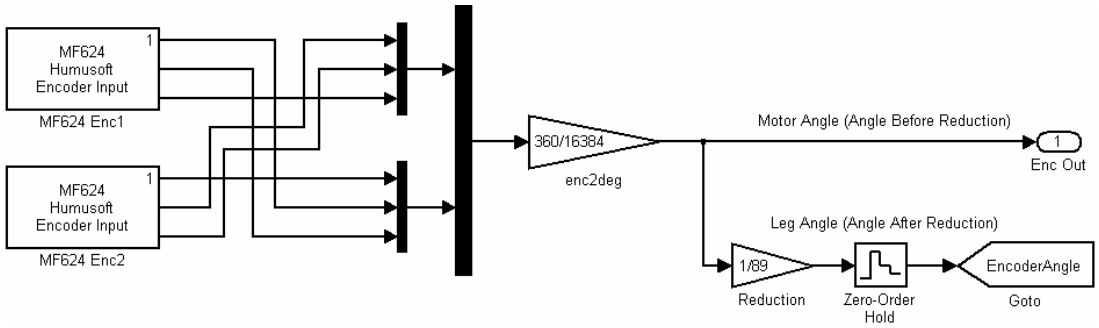

Figure 4.19 : Encoder reading subsystem.

4.5.3 Controller implementation

A feedback control law was designed in Section 3.2 that drives the Hexapod Platform to a desired position. PID (proportional-integral-derivative) controller is used to achieve motion profile of the stabilized head mirror on mobile (top) platform. The following figure shows the implemented controller simulink block.

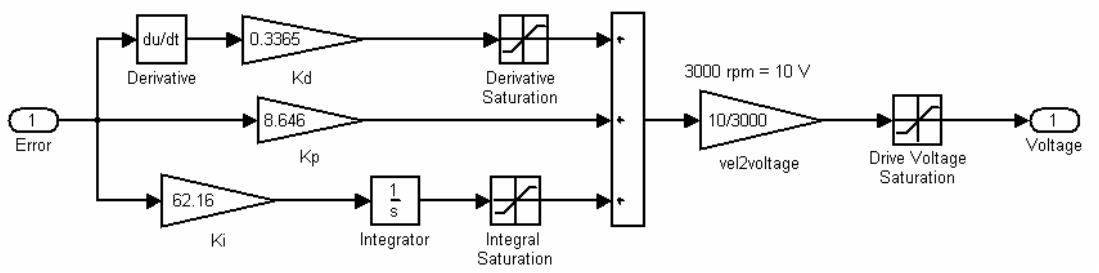

Figure 4.20 : PID controller subsystem.

The PID controllers are identical for each leg. The controllers produce velocity control signals that have to be converted to corresponding voltage demands to drive the servo amplifiers in analog-speed mode. The rpm is directly proportional to the voltage. 10 volt corresponds 3000 rpm and -10 volt corresponds -3000 rpm at contrary direction. The volt-rpm graph is linear.

Voltage demands for six legs are provided by MF 624 analog output channels. The demands are delivered according to electrical design.

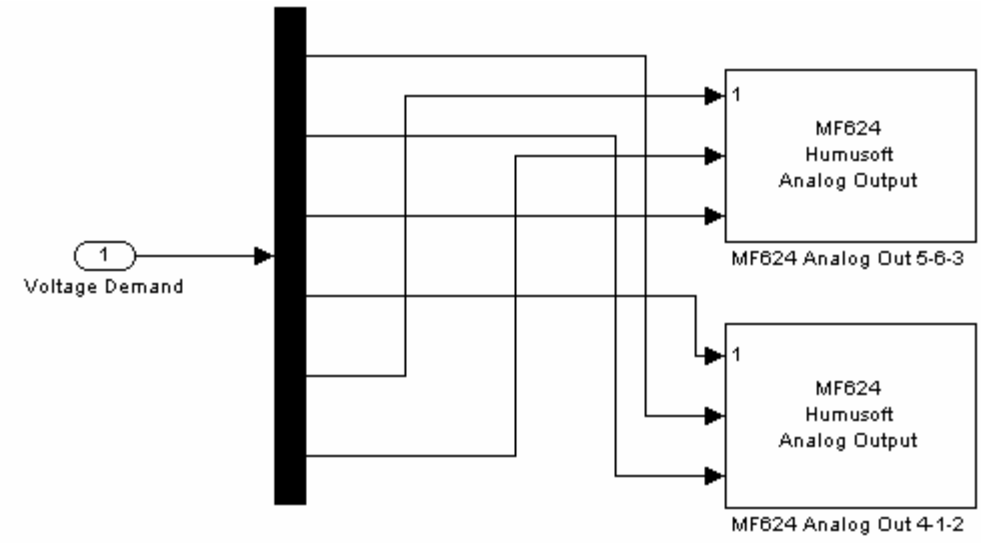

Figure 4.21 : Analog output subsystem.

Adjustment of reference signal gain according to motion demand is one of the auxiliary methods to equalize the reference motion and the actual motion on the mobile platform. xPC Target supports the modification of parameters in the Simulink blocks (setpar <(index_number in [project_name+pt.m] file)-1> = <new value>) while the application is running. The parameter changes are immediately reflected in the real-time application. The tight integration between MATLAB, Simulink, Real-Time Workshop, and xPC Target makes it possible to write a script that incrementally changes a parameter and monitors a signal output to optimize the value of the parameter. The following figure clarifies the iterative method of gain adjustment.

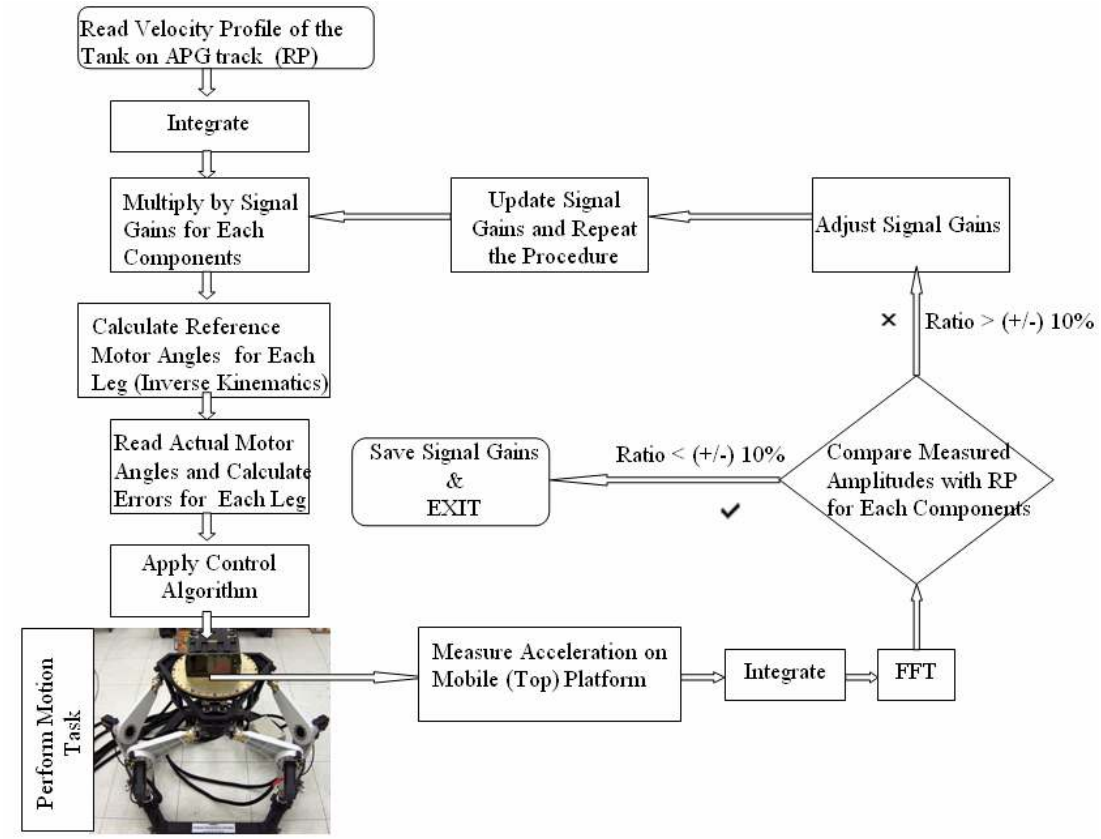

Figure 4.22 : Adjustment of reference signal gain.

PID controller makes the reference motion close to the actual motion. However, some small differences exist at different frequencies. To remove tiny differences between the reference motion and the actual motion, reference signal gains for each components need to be adjusted. The above method is used for this purpose. The accelerometer is located on the mobile platform for each axis. Velocity data can be extracted from acceleration measurement. FFT (Fast Fourier Transform) is applied to velocity data for observation of amplitudes for each frequency. Reference signal gain is adjusted by comparing amplitudes of the reference velocity profile and the measured velocity profile. The following tables show the reference signal gains for each components.

Table 4.3 : Reference signal gains for pitch axis.

Component Number	1	2	3	4	5	6	7	8
Frequency (Hz)	1	2.5	5	10	17	23	30	40
Signal Gain	0.92	1	1.0475	1.2534	1.65	0.94	0.5	3.05

Table 4.4 : Reference signal gains for yaw axis.

Component Number	1	2	3	4
Frequency (Hz)	1	2.5	5	10
Signal Gain	0.94	0.962	1	1.85

All calculation for the mentioned method is done with MATLAB. Reference signal gain results have to be applied in Simulink model. The following figures show the modified pitch and yaw motion subsystems.

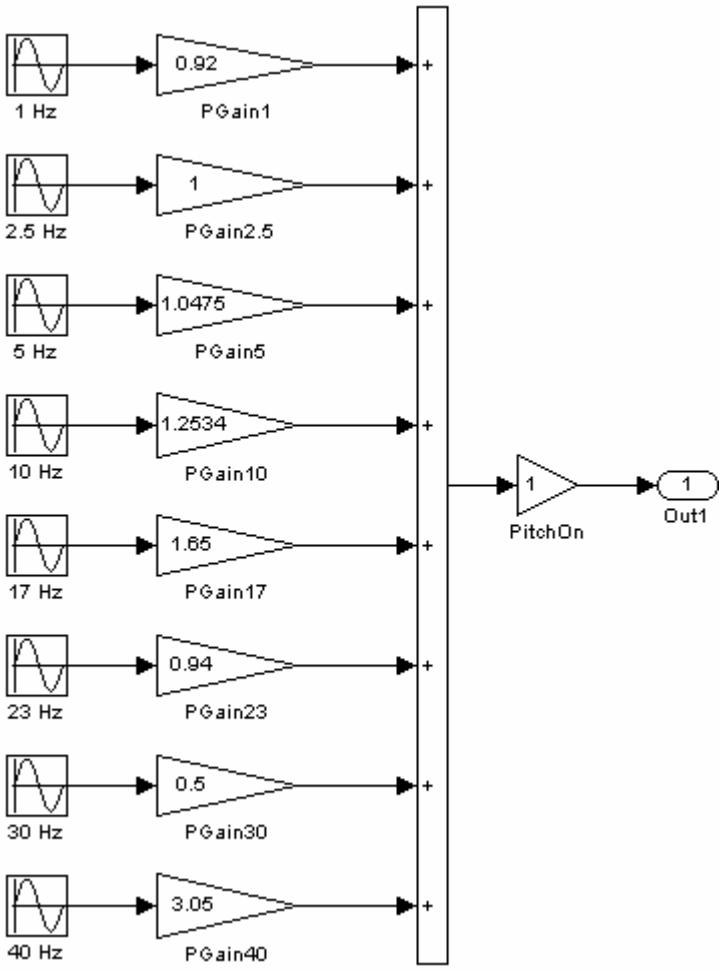

Figure 4.23 : Adjusted pitch motion.

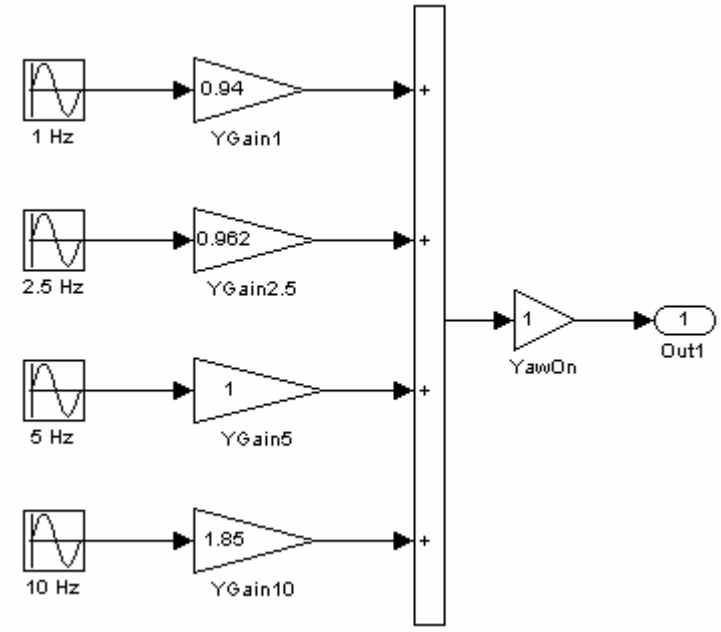

Figure 4.24 : Adjusted yaw motion.

4.5.4 Real-Time computing

Real-time behavior is inherent to embedded systems design. There are different definitions of real-time. It can be defined as a fast enough response for a particular application. Powerful microprocessor is required to achieve real time behavior. The following figure provides a high-level view of the real-time executable.

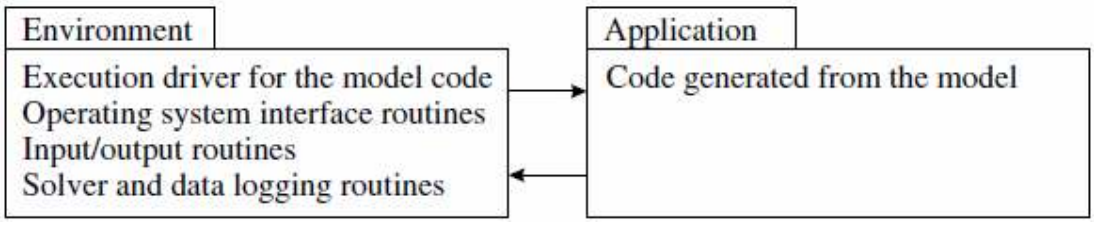

Figure 4.25 : The common view of a real time program.

Measurement from the sensors, creating platform motion demand, inverse kinematics solution, generating and delivering control output signals to the drivers are done by MATLAB® in real-time. Interrupt handling and polling is supported for real time applications by the xPC target embedded option kernel.

4.5.5 Generating embedded code

A typical Simulink block consists of inputs, states, and outputs, where the outputs are a function of the sample time, the inputs, and the block states. During simulation, the model execution follows a series of steps (shown in the following figure). The first

step is the initialization of the model, where Simulink incorporates library blocks into the model; propagates signal widths, data types, and sample times; evaluates block parameters; determines block execution order; and allocates memory. Simulink then enters a simulation loop. Each pass through the loop is referred to as a simulation step. During each simulation step, Simulink executes each of the model blocks in the order determined during initialization. For each block, Simulink invokes functions that compute the values of the block states, the derivatives, and the outputs for the current sample time that is (1/1800) 556 microseconds. The simulation is then incremented to the next step. This process continues until the simulation is stopped.

Processing of the application tasks, communicating with I/O device, and updating the discrete states determine the minimum sample time of the application. Our RT application runs with the minimum sample time, 556 microseconds, to be able to do tasks properly. The application displays error message "ERROR: CPU OVERLOADED AT TIME" on the target' s screen and stops the simulation if the simulation loop is not completed for the current sample time. The minimum sample time of the HIL application is determined as 556 microseconds by trial and error method.

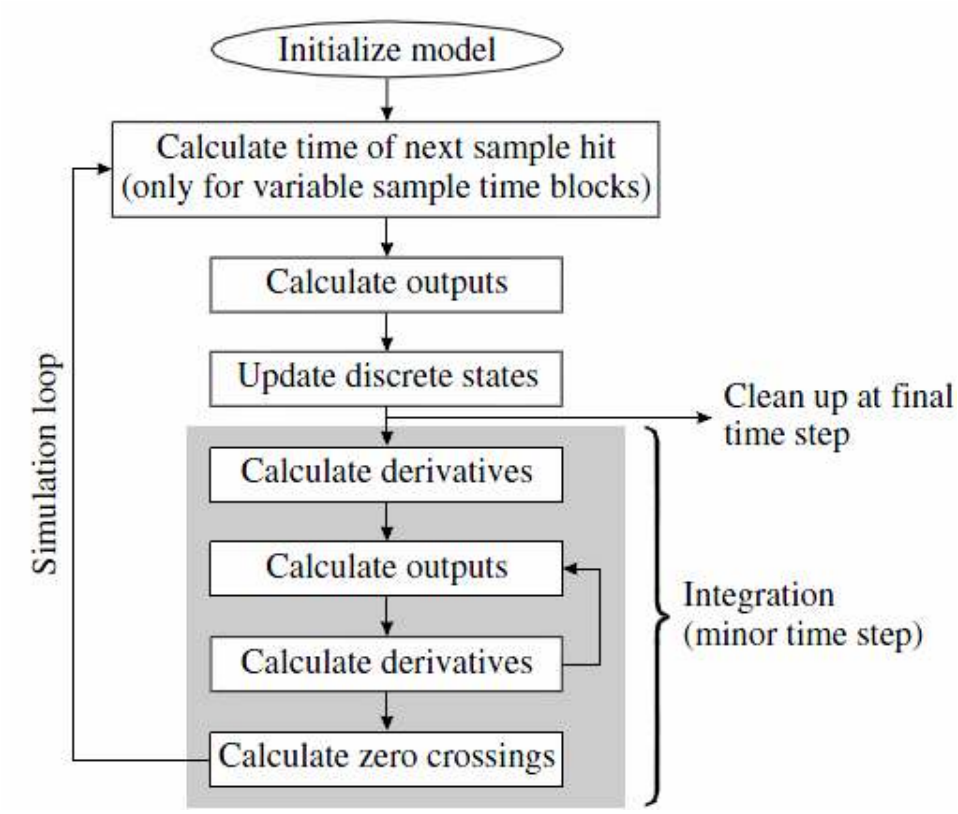

Figure 4.26 : Simulink simulation steps.

MATLAB Real-Time Workshop takes the Simulink model and generates the application or algorithm code that contains the system of equations derived from the model as well as the block parameters and the code to perform initialization. Real-Time Workshop also provides a run-time interface that allows the model code to be built into a complete, stand-alone program that can be compiled and executed.

xPC Target uses this run-time interface and combines it with a real-time clock and scheduler to generate a real-time application, while providing the drivers for interfacing to real-time hardware and the signal monitoring and parameter tuning capabilities. Based on the sample rate specified in the Simulink model, xPC Target uses interrupts to step the execution of the model at the appropriate rate. With each new interrupt, the target application computes all the block outputs from the model, similar to the way Simulink computes its block outputs.

The code generated from the Simulink model is sometimes referred to as the model code because it implements the Simulink model. The model code contains functions that correspond to the applicable simulation steps outlined in the above figure compute the model outputs, update the discrete states, integrate the continuous states (if applicable), and update time. xPC Target generates its own main program. This program interacts with the execution driver for the model code, which in turn calls these functions.

The functions then write their calculated data to the real-time model. At each sample interval, the main program passes control to the model execution function, which executes one-step through the model. This step reads inputs from the external hardware, calculates the model outputs, writes outputs to the external hardware, and then updates the states, as shown in the following figure.

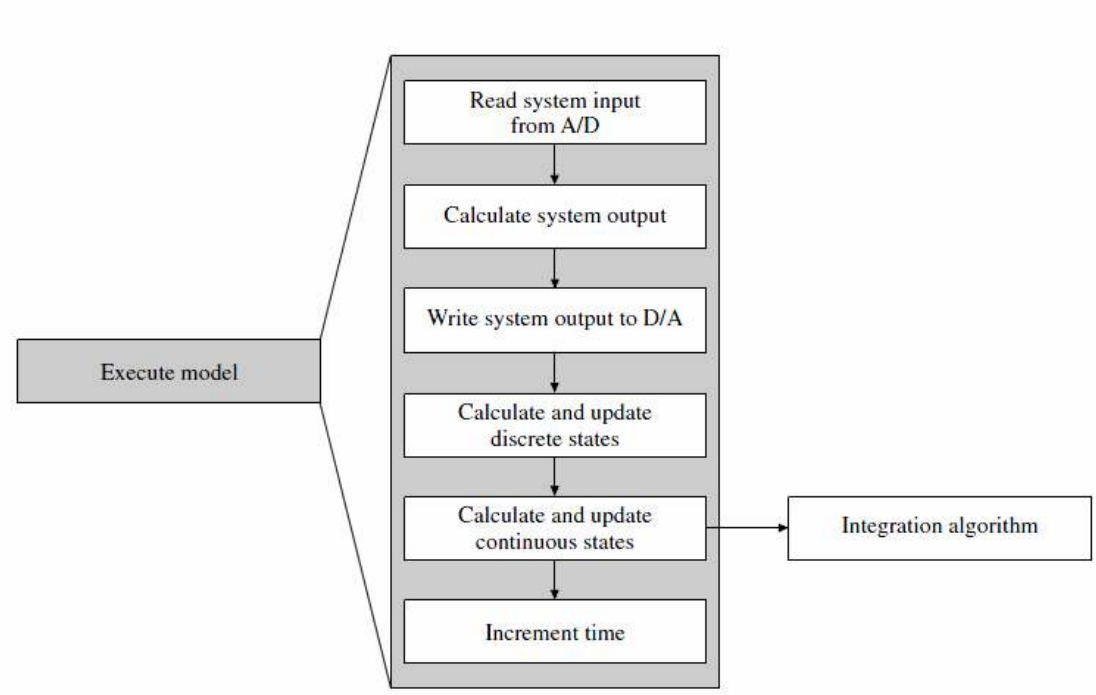

Figure 4.27 : Real-time model code execution.

If these computations require the plant output of the previous sample time, they must be performed in one sample interval. This implies synchronization between the execution of the logical program by the controller and the dynamic behavior of the plant in real time. The sample rate is determined by control law analysis and depends on the time constants of the plant. Note that this scheme writes the system outputs to the hardware before the states are updated. Separating the state update from the output calculation minimizes the time between the input and output operations. The generated code also contains functions to perform initialization, facilitate data access, and complete tasks before program termination. The requirement to have plant input computed at a given point in time implies a fixed controller response time. As a result, the controller cannot rely on iterative computational schemes unless the upper bound of the iterations is fixed. This means that the controller must not employ a variable integration step or include algebraic loops.

The embedded code files for xPC Target Embedded Option are created after building the simulink model. The following figures show the configuration of the parameters.

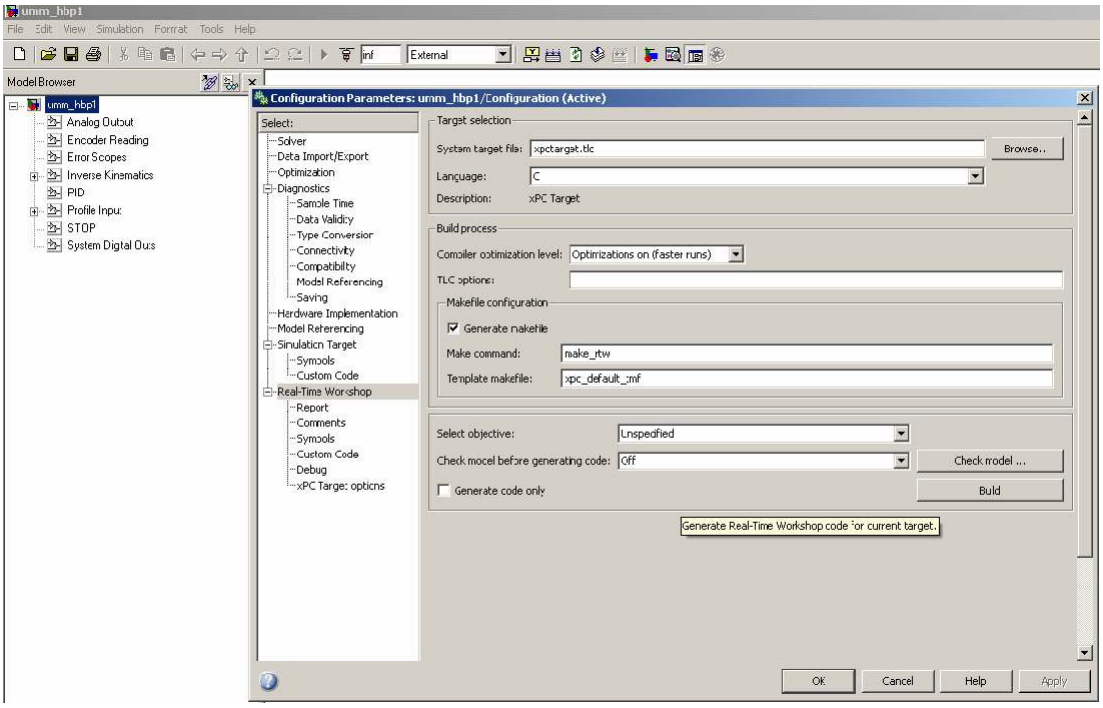

Figure 4.28 : Real-time workshop panel.

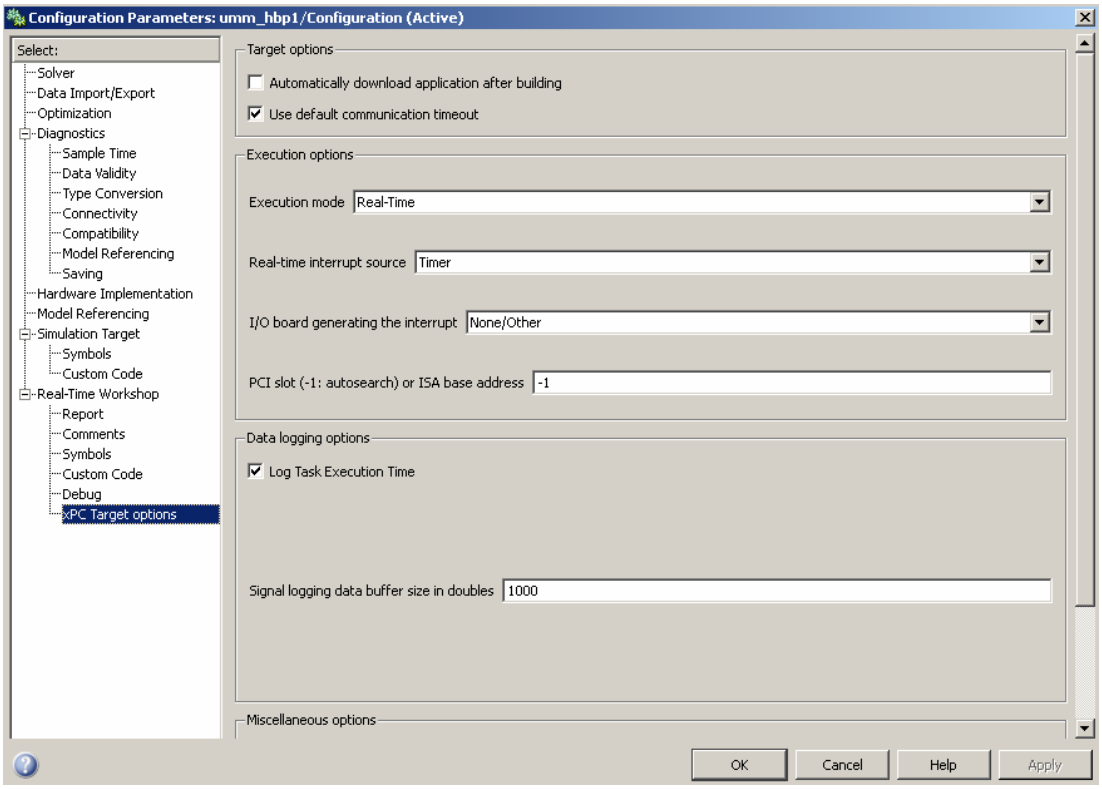

Figure 4.29 : xPC target options panel.

The Standalone mode of xPC Target Embedded Option allows you to use a target PC as a stand-alone system. Standalone mode enables you to deploy control systems, DSP applications, and other systems on PC hardware for use in production

applications using PC hardware [15]. The xPC Target software with Standalone mode can be used to create a combined kernel and target application with utility files created in directory named as modelname_xpc_emb after building the Simulink model. These files are:

- *.rtb, which contains the xPC Target kernel and the target application;
- xpcboot.com, which executes loads and executes the *.rtb file;
- autoexec.bat, which is automatically invoked by DOS runs xpcboot.com and the *.rtb file;
- DOS files, which boot the target PC.

You initially boot your target PC with DOS from any boot device, then the xPC Target kernel is started from DOS. The xPC Target software only needs DOS to boot the target PC and start the xPC Target kernel. DOS is no longer available on the target PC unless you reboot the target PC without starting the xPC Target kernel. The real-time application should now be running on the target PC. The interested reader is referred to MATLAB xPC Target help file for further information.

4.5.6 Drivers

An important step in implementing software into a real time system is the requirement to have device drivers that communicate between the I/O devices (MF 624 PCI boards in our case) on the target PC and the application code running on it. Drivers provide interaction between the real time application and the real physical system. The device driver includes the code that runs on the hardware of the target PC for interfacing to I/O devices such as digital signals, ports, encoders, timers and counters, analog to digital A/D converters and digital to analog D/A converters. Each device driver is implemented as simulink s-function using C code.

4.5.6.1 Writing device drivers

Understanding s-functions and low level programming of I/O boards is fundamental to work on writing device drivers. An s-function is a user definable simulink block that can be written in C, M (level-1), FORTRAN, and C++ and must conform to s-function standards [24]. S-function has a special calling syntax, referred to as a call-back that allows these custom blocks to interact with simulink in the same manner as built in simulink blocks do. S-function can be compiled and dynamically linked into

the simulink environment. Also, s-function methods from an application program interface (API) allow the flexible implementation of generic algorithms within the simulink environment. This flexibility can not always be maintained when s-functions are used with Real-Time Workshop to generate code. For example, it is not possible to access the MATLAB workspace from an s-function that is used with Real-Time Workshop, but it is possible when using the s-function with Simulink for simulation purposes only.

To incorporate a device driver block in the simulink model requires an s-function, which in turn requires the C source code for the device driver. xPC Target provides a comprehensive device driver library supporting many boards of various types, including A/D, D/A converters, encoder inputs, digital inputs/outputs, PWM outputs, signal generators, ethernet, GPIB, MIL-STD 1553, RS 232, Controller Area Network, ARINC 429 etc. Thus, xPC Target greatly simplifies the process of generating a real-time application by providing the library of device driver s-functions.

The xPC Target kernel has direct virtual to physical address mapping, which means that declaring a pointer at a particular address will lead to bus access at the same physical address. Moreover, the source code for the existing xPC Target device drivers is provided with the product, enabling users to gain familiarity with the way device drivers are implemented.

Device drivers for xPC Target can be developed in one of two ways. Firstly, obtaining the source code for the driver from the hardware manufacturer and porting it to the xPC Target kernel. Secondly, using the register programming manual of the I/O board from the hardware manufacturer to develop a driver from the very beginning.

The xPC Target kernel provides a set of functions for accessing ports and memory, PCI initialization space, and performing time measurements that can be used in an s-function compiled for xPC Target. PCI boards are better than ISA boards for adding I/O functionality to a real-time system because they provide plug-and-play allocation of access, interrupt resources, a wider and faster bus, and software calibration.

Also, the developer can add more functional or safer codes to the source code for the driver from the manufacturer. The used Humusoft MF 624 PCI bus card has C source

codes for s-function block from Humusoft Inc., but while testing the system some additional safety and functionality in code is required. Thus, some modifications are applied in the source code such as PCI device bus configuration in mdlStart method of s-function driver. The following figure shows a part from the modified C code of the s-function driver. Detailed driver codes are given in Appendix C.3.

```
#define MDL_START
static void mdlStart(SimStruct *S)
{

#ifndef MATLAB_MEX_FILE
  uint_T i;
  PCIDeviceInfo pciinfo;
  void *Physical1, *Physical0;
  void **Virtual1, **Virtual0;
  volatile unsigned int *base_chipset;
  volatile unsigned short *base_adc;
  char devName[20];
  int  devId;

  switch ((int_T)mxGetPr(DEV_ARG)[0]) {
    case 1:
      strcpy(devName,"Humusoft MF624");
      devId=0x0624;
      break;
    case 2:
      strcpy(devName,"Humusoft AD622");
      devId=0x0622;
      break;
   }

  if ((int_T)mxGetPr(SLOT_ARG)[0]<0) {
    // look for the PCI-Device
    if (rl32eGetPCIInfo(0x186C,(unsigned short)devId,&pciinfo)) {
      sprintf(msg,"%s: board not present.",devName);
      ssSetErrorStatus(S,msg);
      return;
    }
  } else {
    int_T bus, slot;
    if (mxGetN(SLOT_ARG) == 1) {
      bus  = 2; //0;
      slot = (int_T)mxGetPr(SLOT_ARG)[0];
    } else {
      bus = 2; //(int_T)mxGetPr(SLOT_ARG)[0];
      slot = (int_T)mxGetPr(SLOT_ARG)[1];
    }
    // look for the PCI-Device
    if (rl32eGetPCIInfoAtSlot(0x186C,(unsigned short)devId,(slot & 0xff) | ((bus & 0xff)<<
8),&pciinfo)) {
      sprintf(msg,"%s (bus %d, slot %d): board not present.",devName,bus, slot );
      ssSetErrorStatus(S,msg);
      return;
```

Figure 4.30 : A part from MF 624 driver code.

4.5.7 Design of user interface

A windows application with a graphical user interface (GUI) was developed for performing the Hexapod Platform control actions such as amplifier setup, motion start/stop, homing, logging events. The program also acts as a guide for operating the system safely.

The RT engine on the computer with xPC target does not provide a user interface for these tasks. Communication protocols can be used to provide a user interface on the

host computer for RT target application. The host computer displays the GUI of the application while the RT engine executes the embedded code generated by MATLAB Simulink Real-Time Workshop. The user interface thread handles the communication between LabWindows/CVI and xPC target application. The following graph illustrates the main idea of the task processing of the Hexapod Platform.

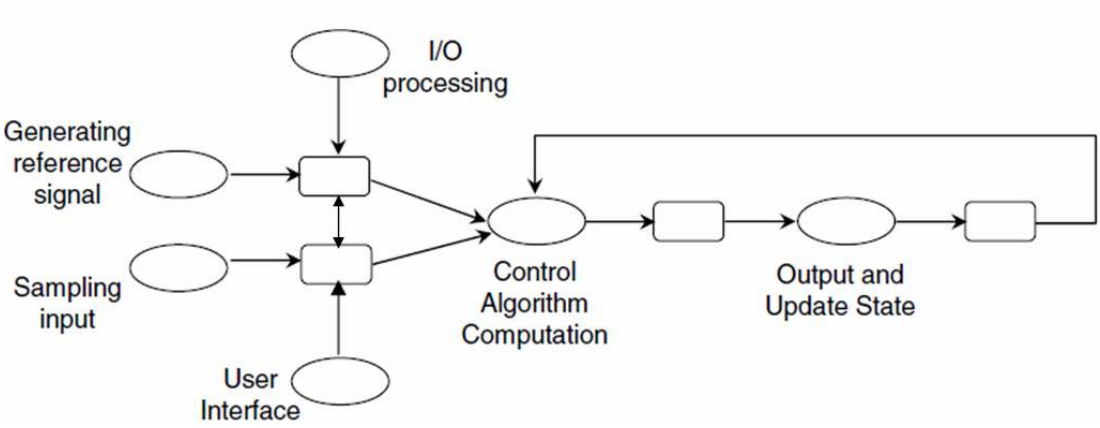

Figure 4.31 : Task graph of the motion simulator.

Using communication protocols is a quick method for monitoring and interfacing with application running on an RT target. However, GUI on RT computer is not deterministic and can affect the determinism of a time-critical application.

Communication methods should be used to increase the efficiency of the application on the xPC target.

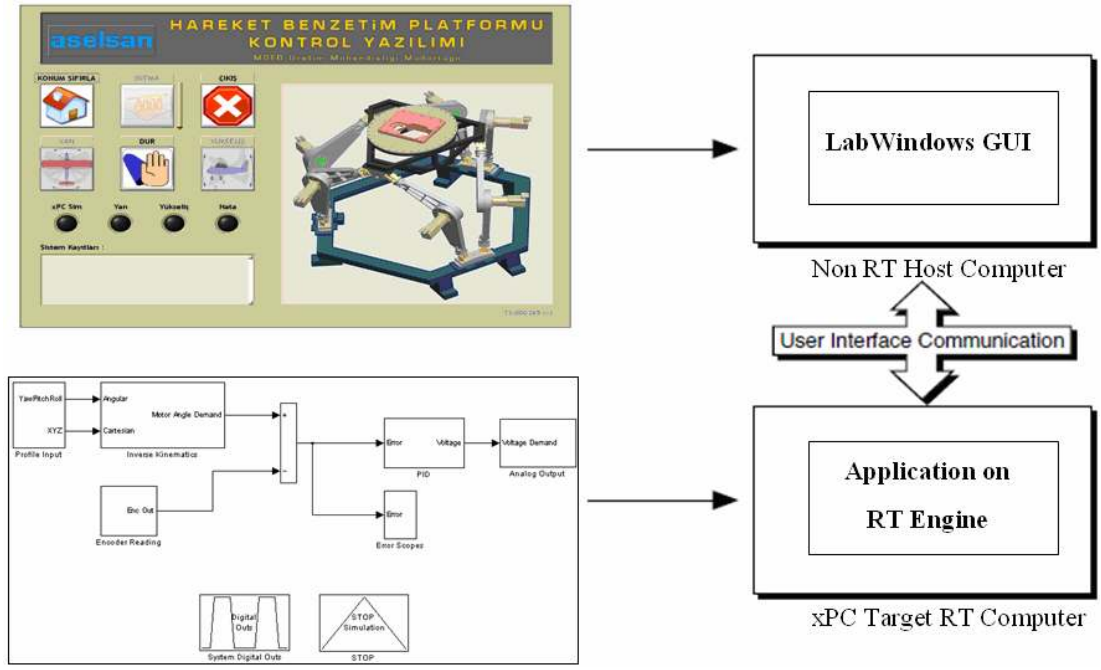

Figure 4.32 : GUI-RT application interaction.

The xPC Target application Simulink model shown in Figure 4.13 runs on real time kernel. The GUI runs on host computer. The GUI communicates with servo amplifiers, electrical control panel, xPC Target Application to achieve the above tasks. The GUI was developed in LabWindows/CVI that is a proven ANSI C integrated development environment that provides engineers and scientists with a comprehensive set of programming tools for creating test and control applications. LabWindows/CVI combines the longevity and reusability of ANSI C with engineering-specific functionality designed for instrument control, data acquisition, analysis, and user interface development [25]. The powerful user interface capability of the LabWindows/CVI can be used to create control design applications for a specific problem or class of problems that enables a repeatable, documented design process.

A LabWindows/CVI Graphical User Interface (GUI) can consist of panels, command buttons, pull-down menus, graphs, strip charts, knobs, meters, and other controls and indicators. You can create a GUI programmatically using function calls, or you can build a GUI in LabWindows/CVI interactively using the User Interface Editor, drop-and-drag editor with tools for designing, arranging, and customizing user interface objects. When you finish designing your GUI in the User Interface Editor, save the GUI as a user interface resource (.uir) file.

The following figures show the designed user interface of the Hexapod Platform.

Figure 4.33 : Connection and initialization user interface.

The above panel sets connection between the host computer, the target computer and the servo amplifiers. It opens communication protocols. Then it checks the servo drivers ready for operation. If a fault presents on any servo, the software aborts the operation and gives the fault message and the responsible staff. If there is not any fault, it sets the system ready to operate.

Figure 4.34 : Control panel of the Hexapod platform.

The motion simulator control software has two threads. The first one is the main thread that collects and responses user interface events (called callbacks in LabWindows). The second one is the system event polling thread that checks the motion simulator hardware status such as servo amplifiers, xPC target process, security switches, excessive resolver errors and so on.

“Konum Sıfırla” button proceeds the homing operation. It sets all servo drivers to digital position mode. Then, all six legs jog towards zero pulse switches on each upper leg and go to the initial position (marked position 120 degrees between lower and upper leg) by jogging in defined number of steps after touching the switch. Finally, the software sets all servo drivers to analog speed mode and gives control of the motion simulator to RT application on xPC Target. RT application proceeds the HIL model by communicating with control panel of the Hexapod Platform. The test profile buttons (Pitch, “Yükseliş”, and Yaw, “Yan”) do not work unless homing is done. Also, the indicators (Simulator in Progress, “xPC Sim” , Yaw Profile in Progress, “Yan”, Pitch Profile in Progress, “Yükseliş”, Error Occurred, “Hata”) are regularly updated according to the system events. All the system event logs are

saved as acrobat document, also displayed in “Sistem Kayıtları” text box. “Dur” button stops the test motion at initial position.

The following figure shows the test flowchart of the stabilized head mirror.

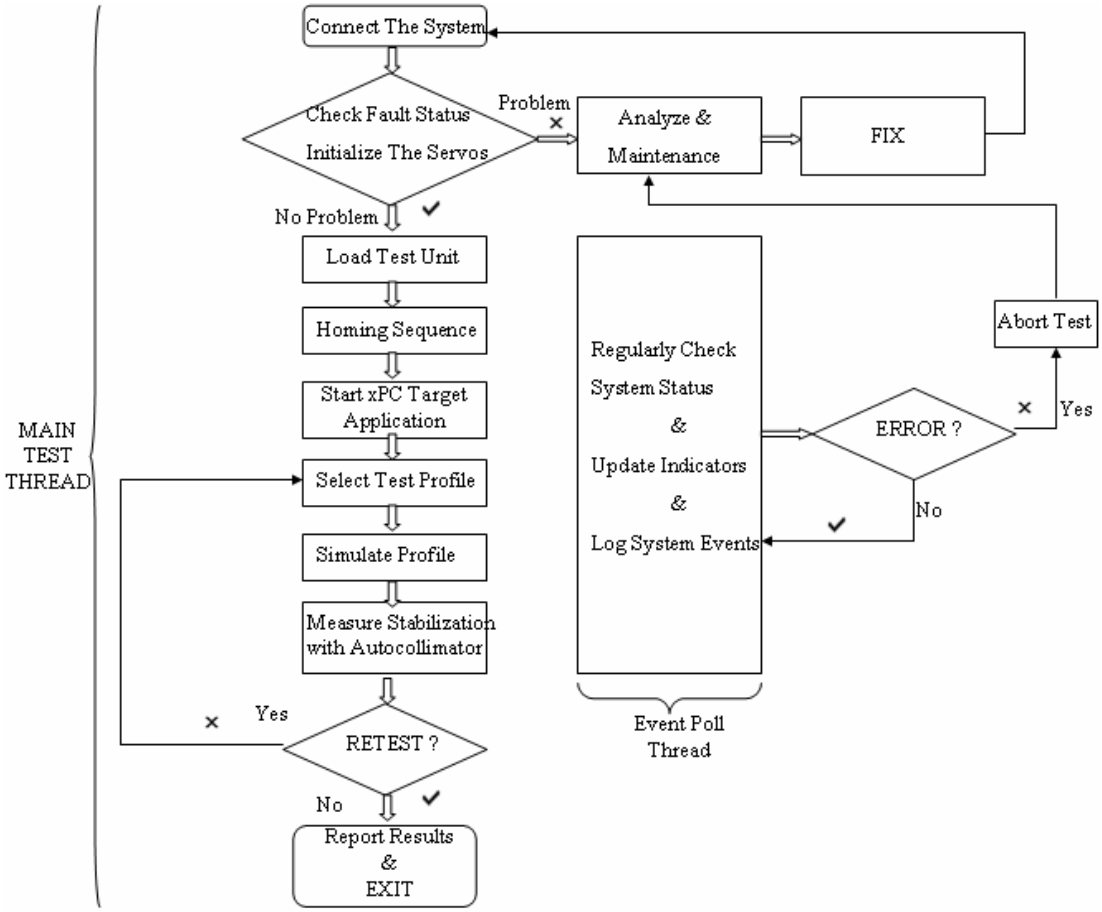

Figure 4.35 : Stabilized head mirror test flowchart.

4.5.8 Safety

Embedded control systems are fail-critical, and they exhibit potentially dangerous behavior to the point where they may trigger catastrophic events. For this reason, the legs on the Hexapod platform must not be driven beyond their maximum extension. This condition is ensured by a control that enforces a hardware limit (shown in “Security Switches on a Leg” figure). The controller uses the current extension of each leg and the requested force to be exerted upon this leg. If a leg moves greater than 30 degrees in positive or negative direction from the initial position, the amplifier will not allow further extension and will brake the motor. Also, the real-time application will lock the motor position, and be stopped automatically if the error between desired position and actual position is greater than 20 degrees on any

motor. The errors on each leg are displayed in real-time on the screen of xPC target computer as shown in the following figure.

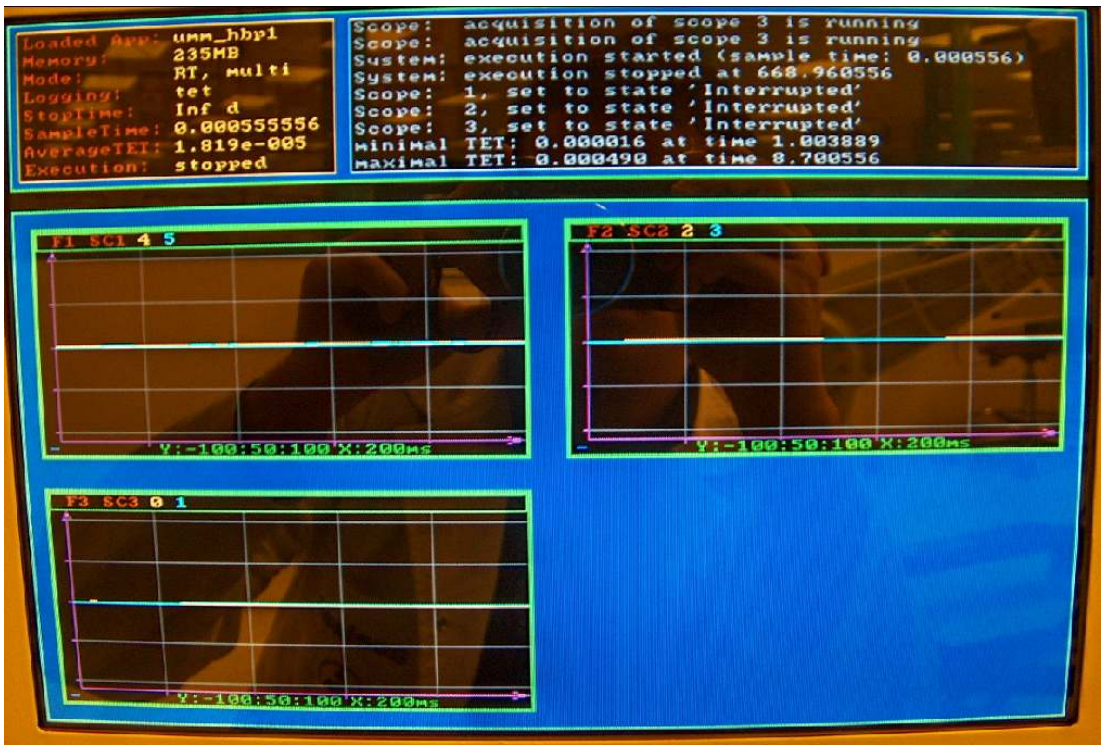

Figure 4.36 : Real-time computer application screen.

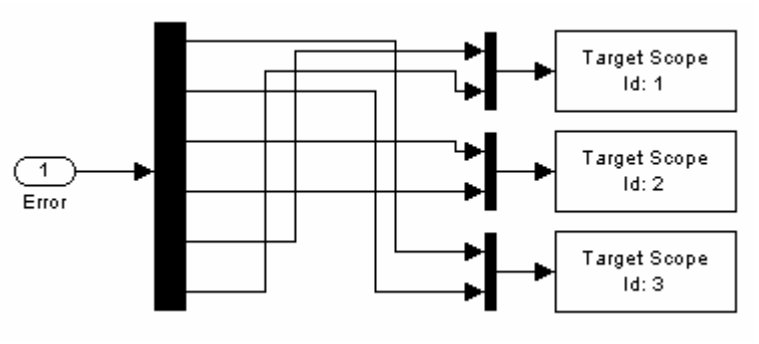

Figure 4.37 : Error scopes subsystem.

In many cases, emergency hardware is available that allows a safety switch to immediately invoke a safe controller. Often, this safe controller is nothing but a simple strategy to shut off power (the big red button named Emergency Stop that is shown in “Electrical Control Panel” figure). After pressing Emergency Stop button of the Hexapod platform, the real-time application is stopped, all motors are locked at the current position, and the control panel gives visual and auditory alarm.

Moreover, it is recommended that additional safety scenarios should be applied at the hardware driver level such as initialization and finalization of the hardware. For this purpose, s-function driver of MF 624 is modified.

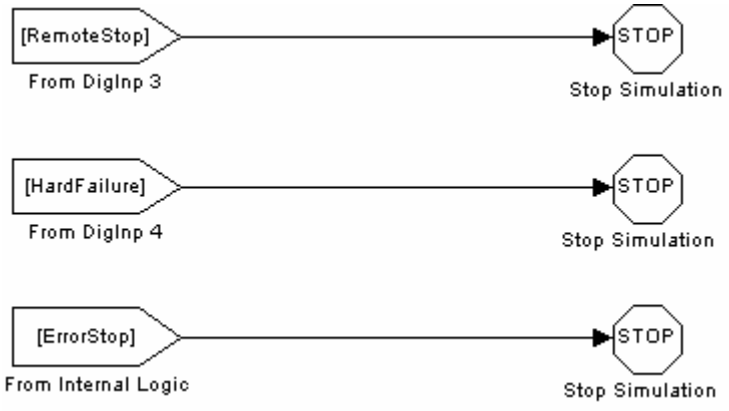

Figure 4.38 : Stop simulation subsystem.

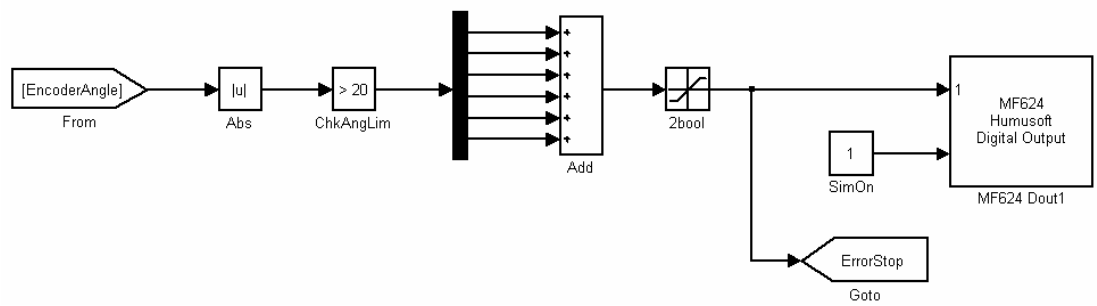

Figure 4.39 : System digital outs subsystem.

4.5.9 Servo drive interface of the amplifiers

Drive.exe is an axis-commissioning tool for both single-axis and multi-axis motion control applications. With its graphical user interface and Windows dialogues, Drive.exe provides an easy point-and-click method for configuring servo amplifiers [26].

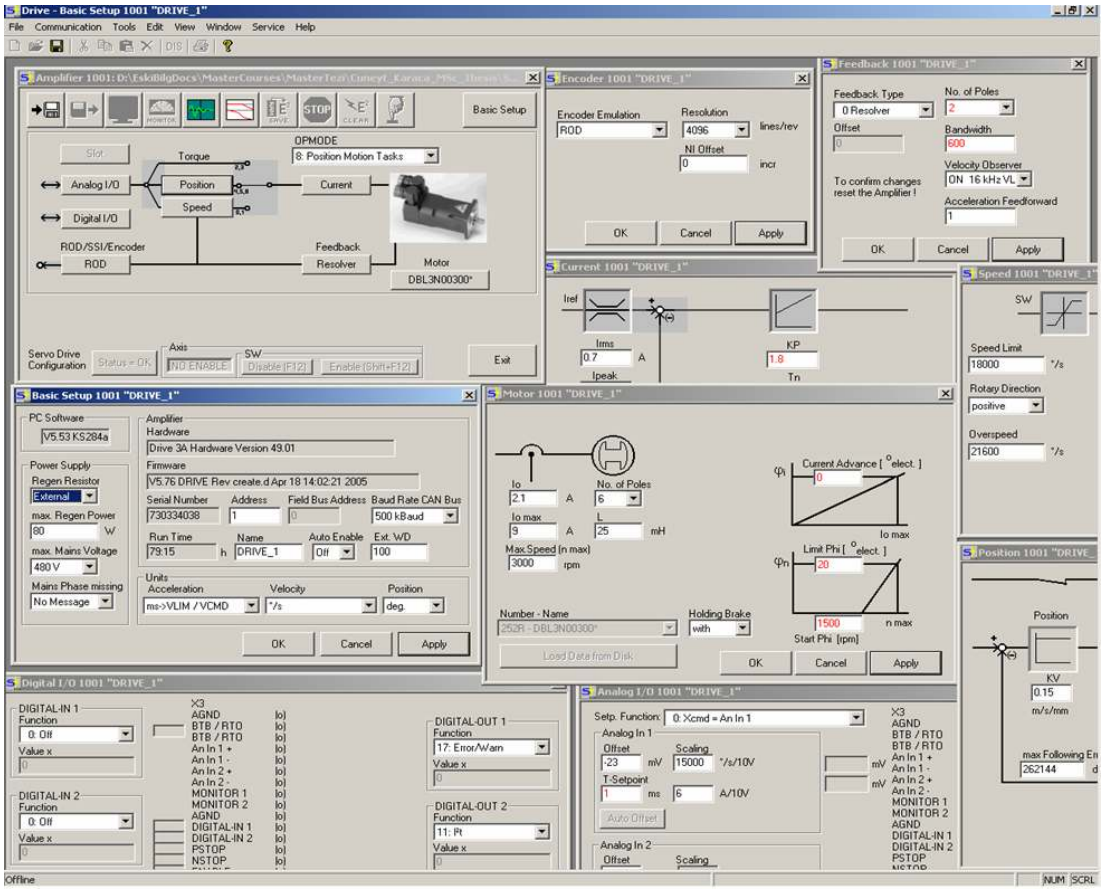

Figure 4.40 : Driver software screen layout.

During the configuration process, the software allows you to tune (stabilize) the servomotor for each axes quickly and efficiently. From Drive.exe, while online with an axis and its motor, you adjust servo parameter values (such as gains and limits) and execute them immediately. While watching and listening to the motor spin, you may use the Drive.exe oscilloscope to adjust and readjust these values until the motor reaches its best performance - optimum speed without oscillation. The changes made to the servo parameter values should be saved to the servo amplifier or to a file.

The dialogues of Drive.exe software step you through the complete startup phase of your programming projects. All parameters in the servo drive can be saved to a separate file for each axis. Each drive file is a unique custom configuration for that drive and can be accessed offline (not connected to the drive) or online (connected to the drive). The six drive files for each servo amplifier are configured and saved to disk. Finally, the currently valid parameters in configuration of each amplifier are downloaded into the EEPROM of the related servo amplifier. In this way, all the parameters are permanently saved. The interested reader is referred to the help file of Drive.exe by Danaher Motion for further information.

4.6 Stabilized Head Mirror Test Equipments

Extra test equipments are required beyond the motion simulator to test the stabilized head mirror (SHM). These are control panel compatible with hardware and software of stabilized head mirror and an autocollimator. The designed system is given in the following figure.

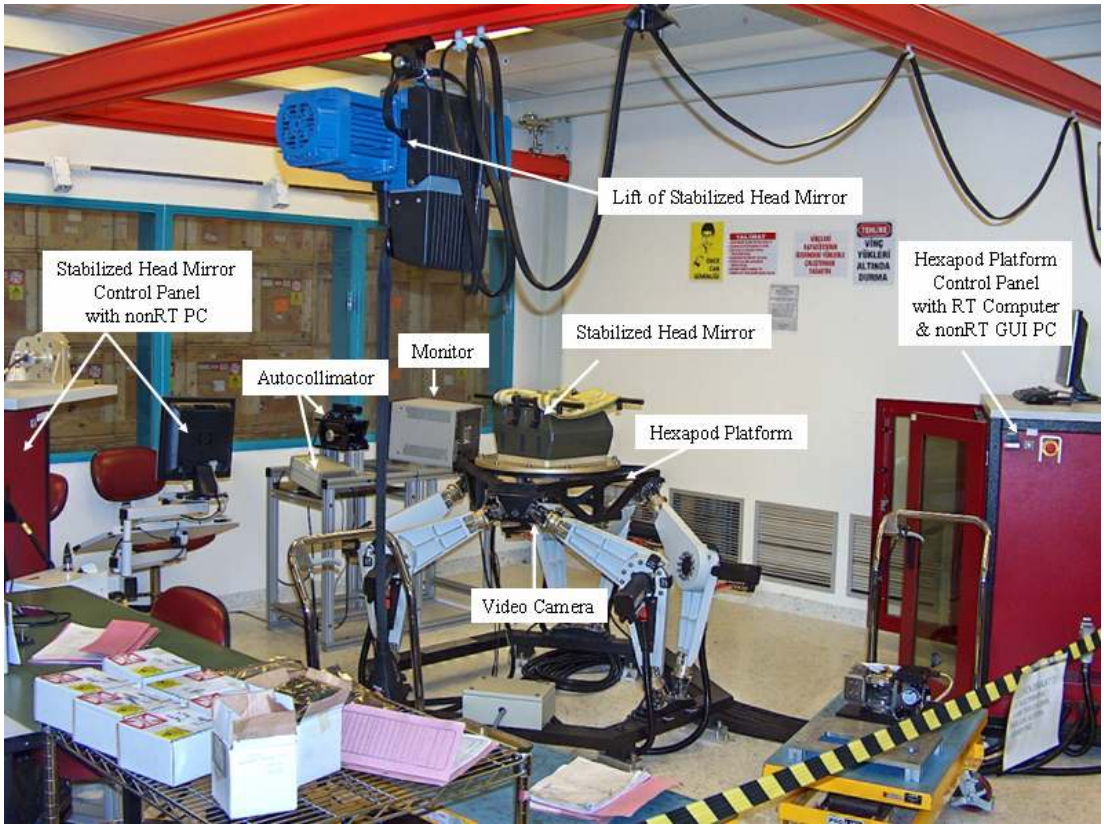

Figure 4.41 : Designed test system.

An autocollimator is an optical instrument that is used to measure small angles with very high sensitivity. As such, the autocollimator has a wide variety of applications including precision alignment, detection of angular movement, verification of angle standards, and angular monitoring over long periods [27]. It will simultaneously measure angular changes about two orthogonal axes. The axes of measurement are commonly referred to as azimuth and elevation or yaw and pitch. The autocollimator projects a beam of collimated light. An external reflector reflects all or part of the beam back into the instrument where the beam is focused and detected by a photo detector. The autocollimator measures the deviation between the emitted beam and the reflected beam. Because, the autocollimator uses light to measure angles, and it never comes into contact with the test surface.

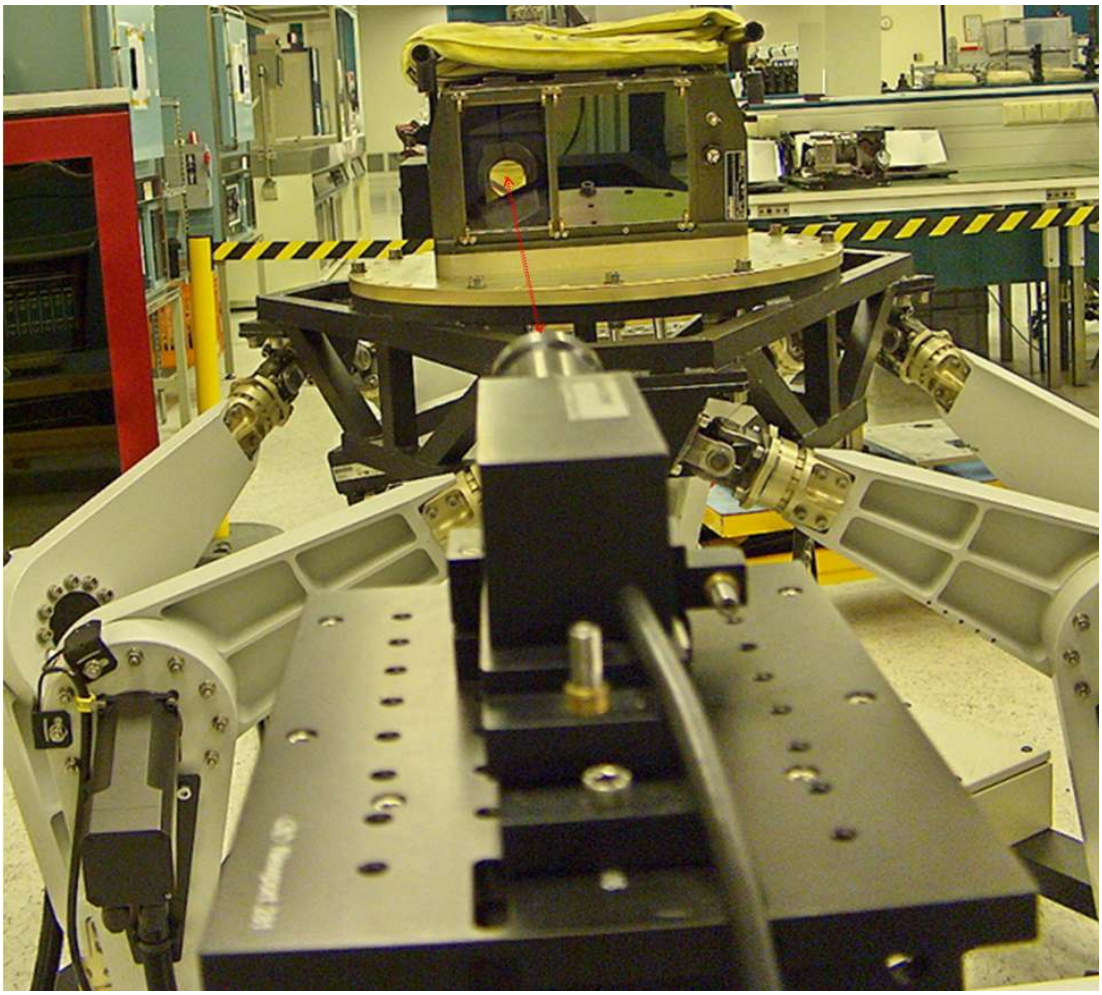

Figure 4.42 : Autocollimator-SHM interaction.

Digital autocollimators use an electronic photo detector to detect the reflected beam. Micro-Radian autocollimators take advantage of the latest detector technology including advanced silicon-based photo detectors and germanium-based detectors. The detector sends a signal to the Micro-Radian E3 digital controller, which digitizes and processes the signal using proprietary DSP-based electronics. The processing creates a calibrated angular output, which is traceable to the U.S. National Institute of Standards and Technology. The angular data is retrieved using the digital LCD display, the RS-232 interface, or the analog outputs, which are all, built into the E3 controller. Digital autocollimators are suitable for applications. including calibrating rotary tables, checking angle standards, remote or long term angular monitoring, measurements of flatness or straightness, and to provide angular feedback in servo-controlled systems.

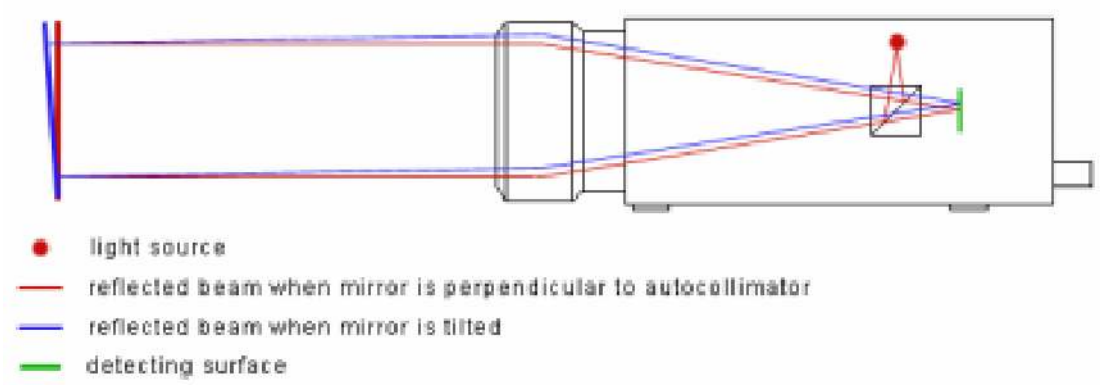

Figure 4.43 : Digital autocollimator diagram.

"SBKB Gözcü" software is the control software of the stabilized head mirror. Settings of SHM can be modified according to predefined interface control document by using this software.

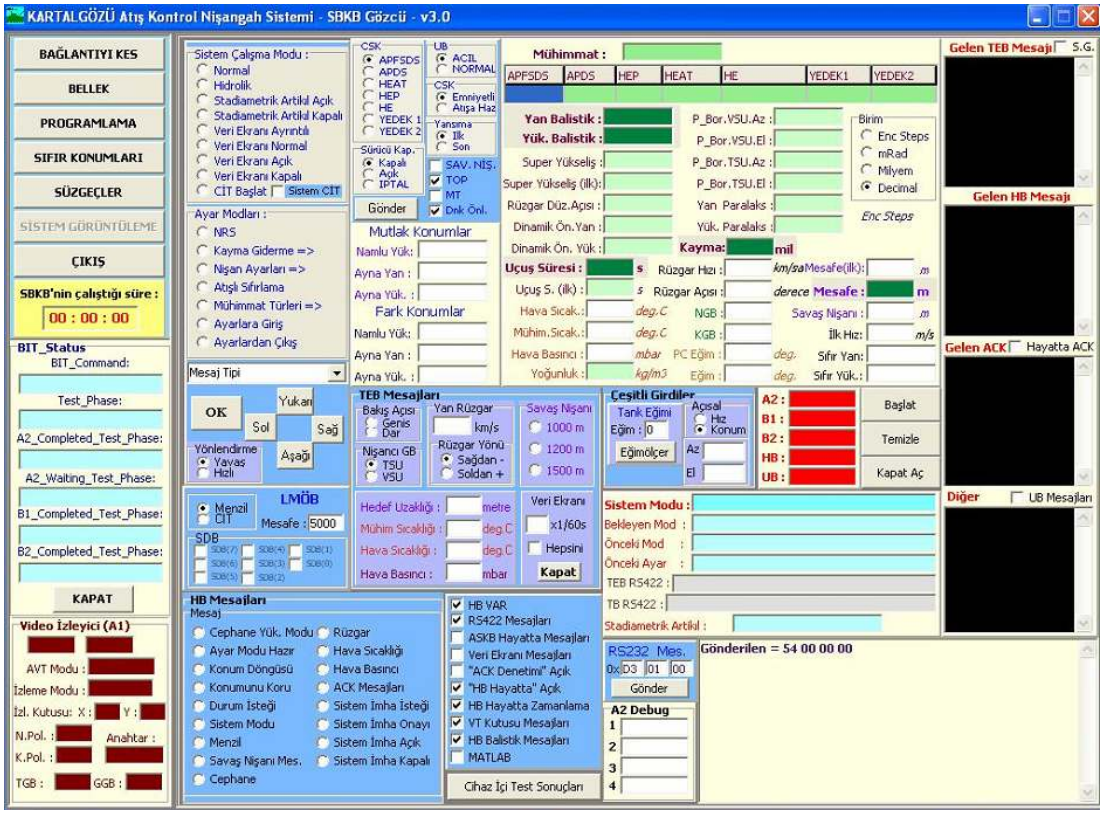

Figure 4.44 : User interface of "SBKB Gözcü" software.

Stabilization test software of SHM has two panels. First one is the main test panel that communicates with autocollimator. It checks the signal level of the collimator light source. Then, if the signal level is greater than 60%, the software displays the deviation measured by autocollimator. The software communicates with E3 digital controller of Micro-Radian TL40 autocollimator over RS-232 protocol. After taking

measurements, the software checks the stabilization limits for yaw and pitch axis. Finally, the test result is saved into the disk and printed out.

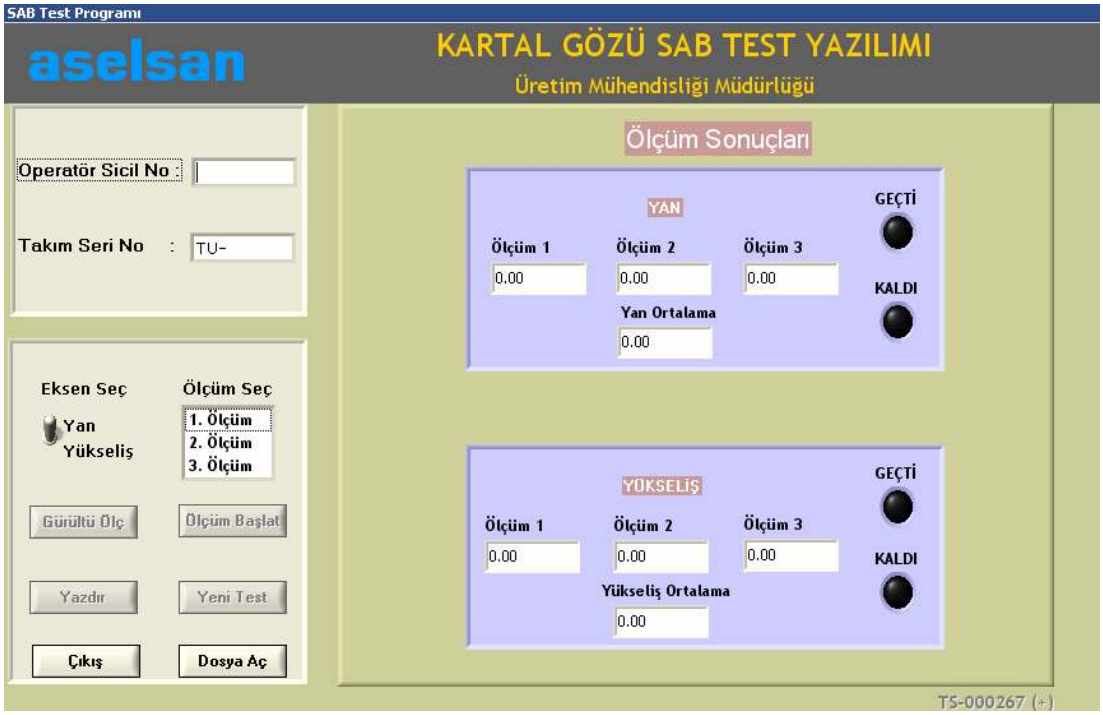

Figure 4.45 : Stabilization test panel of SHM.

Second one is the noise information panel. This study is done to detect noisy signals and spikes on gyro of the stabilized head mirror. The software reads the analog gyro signal by digitizing with MF 624 card. Eventually, abnormal products are determined by the help of the following panel.

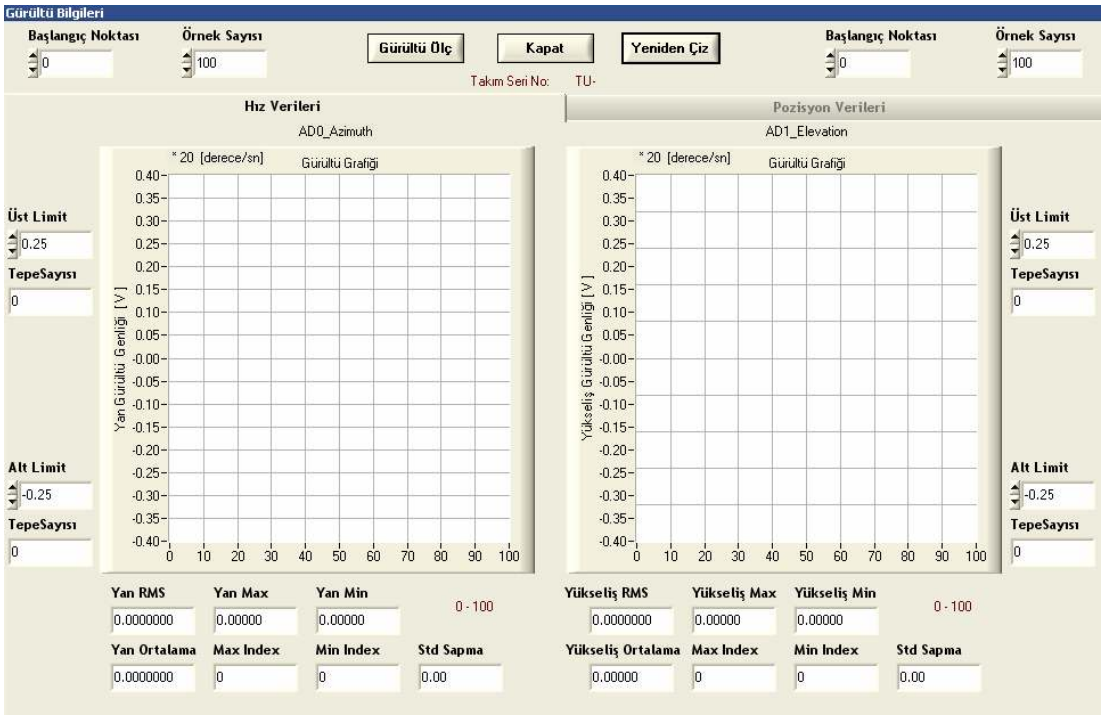

Figure 4.46 : Panel of gyro noise measurement in SHM.

4.7 Verification of the Designed System

We have to check the motion accuracy of the Hexapod Platform. Therefore, comparison between desired motion profile and actual motion measurement is required. For this purpose, three piezoelectric accelerometers are placed on the mobile platform. Then, data acquisition is started with Portable Pulse Type 3560 device by Brüel & Kjær Inc. while the Hexapod Platform is moving for yaw or pitch axis. The Pulse Multi-analyzer software offers a comprehensive selection of analysis methods such as FFT, digital filtering, CPB and other post-processing tools [28]. Finally, the actual velocity measurements on mobile platform for each frequency are obtained after the integration of the acceleration data and FFT processing. The interested reader is referred to Piezoelectric Accelerometers and Vibration Preamplifiers Theory and Application Handbook by Brüel & Kjær Inc. for further information.

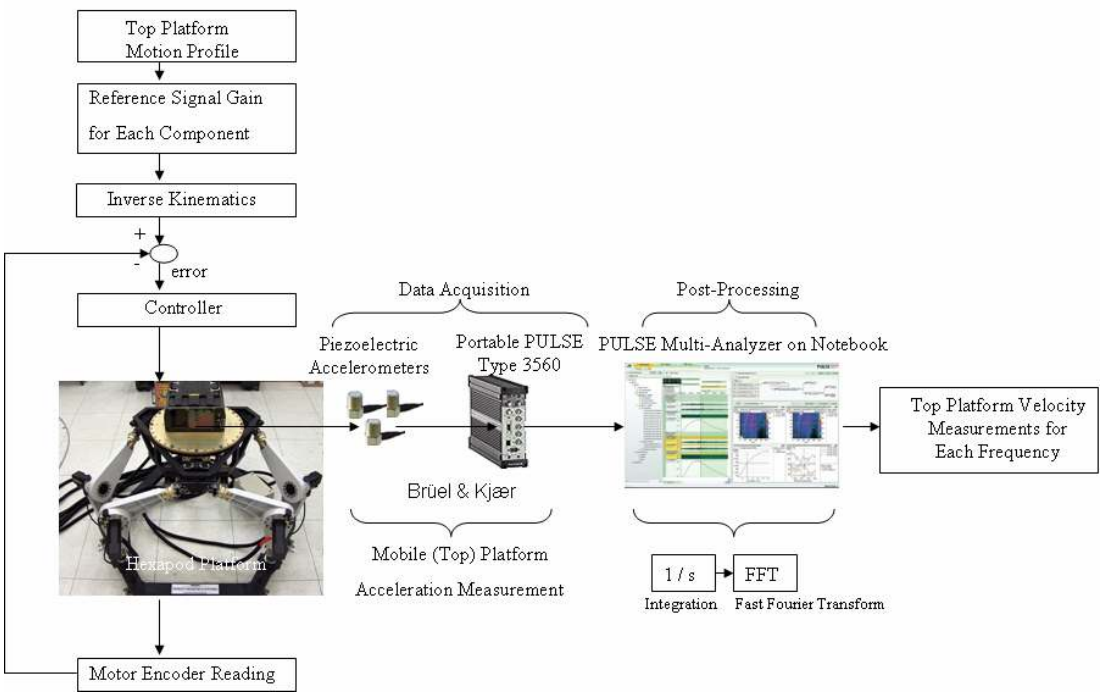

Figure 4.47 : Verification workflow.

Measurements are given in the following figures and tables for pitch axis and yaw axis, respectively. As seen from the tables, the actual profile on the top platform is very close to the demanded profile. The difference is always less than 10% and this is acceptable.

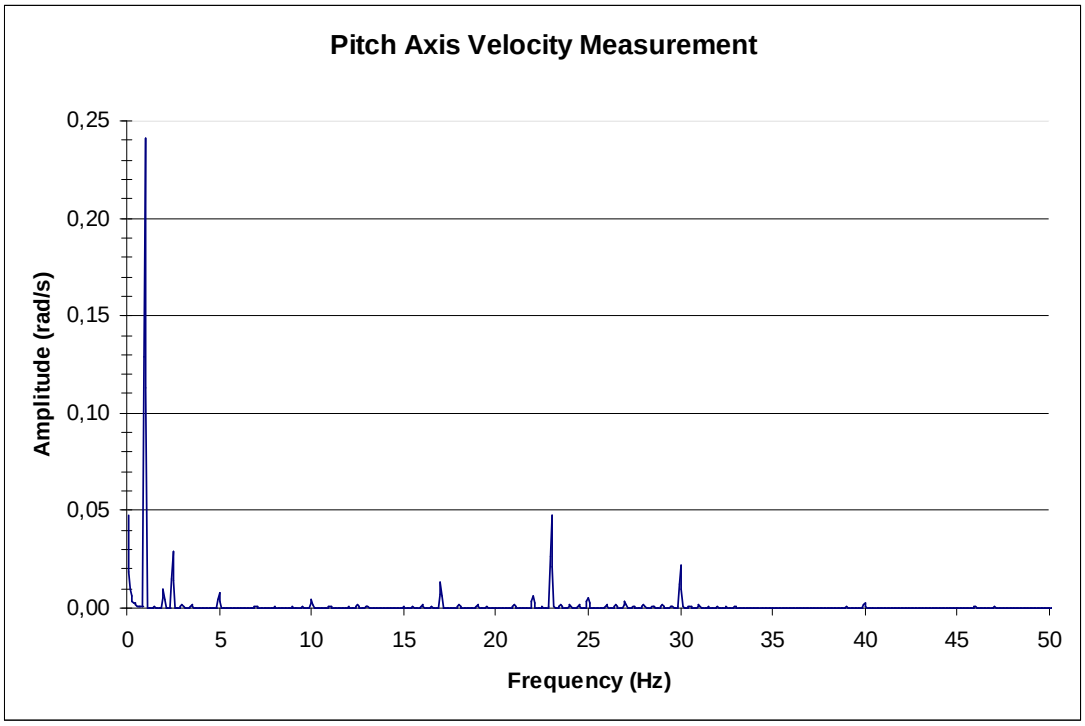

Figure 4.48 : FFT of pitch axis velocity data.

Table 4.5 : Comparison of measured and demanded signals for pitch axis.

Components (Frequency)	Measured Amplitude (rad/sec)	Demanded Amplitude (rad/sec)	Proportion (Measured/Demanded)
1 Hz	0,2409	0,238	1,012
2.5 Hz	0,0289	0,028	1,032
5 Hz	0,0082	0,0084	0,975
10 Hz	0,0045	0,0042	1,071
17 Hz	0,0130	0,0123	1,059
23 Hz	0,0474	0,045	1,053
30 Hz	0,0221	0,021	1,051
40 Hz	0,0031	0,0028	1,096

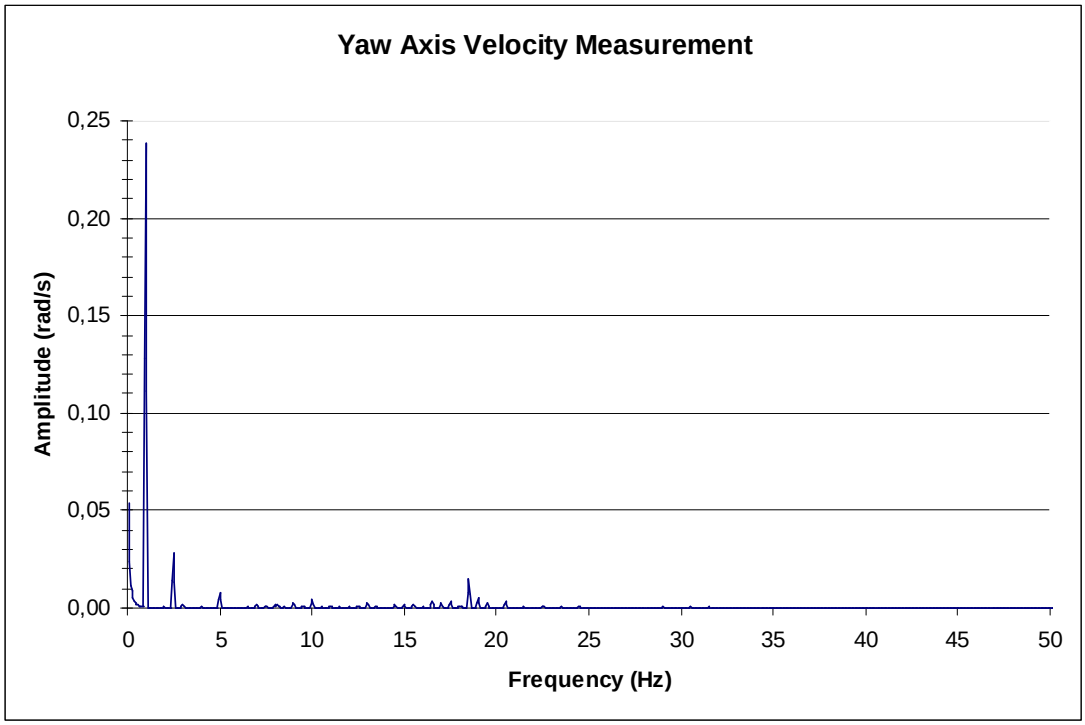

Figure 4.49 : FFT of yaw axis velocity data.

Table 4.6 : Comparison of measured and demanded signals for yaw axis.

Components (Frequency)	Measured Amplitude (rad/sec)	Demanded Amplitude (rad/sec)	Proportion (Measured/Demanded)
1 Hz	0,2385	0,238	1,002
2.5 Hz	0,0279	0,028	0,996
5 Hz	0,0083	0,0084	0,989
10 Hz	0,0043	0,0042	1,017

Verification procedure is repeated five times under supervision of Design Quality Department of ASELSAN. After repeating test five times (the test results are given below), the designed system is approved by Design Quality Department.

Table 4.7 : Verification test results for pitch axis.

Component Number	1	2	3	4	5	6	7	8
Frequency (Hz)	1	2.5	5	10	17	23	30	40
Demanded Amplitude (rad/sec)	0.238	0.028	0.0084	0.0042	0.0123	0.045	0.021	0.0028
Measured Amplitude (rad/sec) # 1 #	0,2410	0,0289	0,0082	0,0045	0,0129	0,0460	0,0222	0,0031
Proportion # 1 #	**1,0126**	**1,0325**	**0,9777**	**1,0602**	**1,0512**	**1,0229**	**1,0552**	**1,1000**
Measured Amplitude (rad/sec) # 2 #	0,2409	0,0289	0,0082	0,0044	0,0130	0,0464	0,0224	0,0030
Proportion # 2 #	**1,0122**	**1,0318**	**0,9732**	**1,0557**	**1,0528**	**1,0311**	**1,0686**	**1,0875**
Measured Amplitude (rad/sec) # 3 #	0,2413	0,0289	0,0082	0,0045	0,0130	0,0470	0,0223	0,0031
Proportion # 3 #	**1,0139**	**1,0321**	**0,9760**	**1,0605**	**1,0561**	**1,0442**	**1,0619**	**1,0982**
Measured Amplitude (rad/sec) # 4 #	0,2411	0,0289	0,0082	0,0045	0,0130	0,0470	0,0223	0,0031
Proportion # 4 #	**1,0130**	**1,0311**	**0,9782**	**1,0610**	**1,0561**	**1,0453**	**1,0605**	**1,1004**
Measured Amplitude (rad/sec) # 5 #	0,2409	0,0289	0,0082	0,0045	0,0130	0,0474	0,0221	0,0031
Proportion # 5 #	**1,0122**	**1,0321**	**0,9748**	**1,0705**	**1,0585**	**1,0529**	**1,0510**	**1,0954**
Mean of Proportions	***1,0128***	***1,0319***	***0,9760***	***1,0616***	***1,0550***	***1,0393***	***1,0594***	***1,0963***

Table 4.8 : Verification test results for yaw axis.

Component Number	1	2	3	4
Frequency (Hz)	1	2.5	5	10
Demanded Amplitude (rad/sec)	0.238	0.028	0.0084	0.0042
Measured Amplitude (rad/sec) # 1 #	0,2384	0,0279	0,0083	0,0044
Proportion # 1 #	**1,0017**	**0,9964**	**0,9887**	**1,0550**
Measured Amplitude (rad/sec) # 2 #	0,2386	0,0279	0,0083	0,0044
Proportion # 2 #	**1,0025**	**0,9961**	**0,9938**	**1,0445**
Measured Amplitude (rad/sec) # 3 #	0,2384	0,0279	0,0083	0,0043
Proportion # 3 #	**1,0017**	**0,9964**	**0,9885**	**1,0257**
Measured Amplitude (rad/sec) # 4 #	0,2384	0,0279	0,0083	0,0043
Proportion # 4 #	**1,0017**	**0,9961**	**0,9920**	**1,0233**
Measured Amplitude (rad/sec) # 5 #	0,2384	0,0279	0,0083	0,0043
Proportion # 5 #	**1,0017**	**0,9961**	**0,9889**	**1,0169**
Mean of Proportions	***1,0018***	***0,9962***	***0,9904***	***1,0331***

As seen from the above tables, the verification test results satisfy the system design specifications.

5. CONCLUSION

Test system design for ASELSAN's stabilized head mirror is discussed with emphasis on motion simulation in this thesis. The Hexapod platform plays a central role in this work, as the use of general-purpose motion simulator with six degrees-of-freedom carries a real world motion into laboratory environment. This makes the Hexapod platform particularly well suited for vibration, displacement, velocity and acceleration test applications. Design and implementation steps of the system that are composed of kinematics of the hexapod platform, modelling the system, controller design and implementation, safety, communication protocols, electrical design, hardware and software implementation, reference trajectory generation, and generating embedded code, are given in the previous sections in detail.

Pro Engineer Wildfire, MathWorks tools (MATLAB, Simulink, SimMechanics, Real-Time Workshop, xPC Target) and LabWindows/CVI software are used during design and implementation stages. These design tools assist in the development of hardware in the loop simulations and improve the capabilities of the system.

The system is finalized with the design of graphical user interfaces that makes the application simpler, more manageable, more functional, and more professional. Lastly, verification of the designed system is done and accepted by ASELSAN Design Quality Department.

Different types of controllers can be designed and implemented for the Hexapod platform as a future scope.

REFERENCES

[1] **Url-1**, < http://en.wikipedia.org/wiki/Serial_manipulator >, accessed 05.08.2009.

[2] **Url-2**, < http://en.wikipedia.org/wiki/Parallel_manipulator >, accessed 05.08.2009.

[3] **D. Stewart,** "A Platform with Six Degrees of Freedom", *Proceedings of the Institute of Mechanical Engineering*, Vol. 180, Part 1, No. 5, pp.371-386, 1965.

[4] **Tanio Tanev,** "Workspace of a Hybrid (Parallel-Serial) Robot Manipulator", *Problems of Engineering Cybernetics and Robotics,* 56, Sofia, 2006.

[5] **D.C.H. Yang and T.W. Lee,** "Feasibility Study of a Platform Type of Robotic Manipulators from a Kinematic Viewpoint", *Journal of Mechanisms, Transmissions and Automation in Design*, 106(2), pp. 191-198, 1984.

[6] **MATLAB®**. *The Language of Technical Computing.* 2008. [online]. [Accessed 2nd September]. Available from World Wide Web: www.mathworks.com/products/matlab/

[7] **Simulink®**. *Simulation and Model-Based Design*. 2008. [online]. [Accessed 2nd September]. Available from World Wide Web: www.mathworks.com/products/ simulink /

[8] **SimMechanics™**. *Model and Simulate Mechanical Systems.* 2008. [online]. [Accessed 2nd September]. Available from World Wide Web: www.mathworks.com/products/ simmechanics /

[9] **D. Craig,** 1996. *Advantages of Simulation.* 2008. [online]. [Accessed 9th September]. Available from World Wide Web: http://www.cs.mun.ca/~donald/msc/node6.html

[10] **SimMechanics™**. 2008. [online]. [Accessed 15th September]. Available from World Wide Web: http://www.mathworks.com/access/helpdesk/help /toolbox/physmod/mech/index.html?/access/helpdesk/help/toolbox/ physmod/mech/f14-6010.html

[11] **SimMechanics™**. *Key Features.* 2008. [online]. [Accessed 15th September]. Available from World Wide Web: http://www.mathworks.com/ products/simmechanics/description1.html.

[12] **SimMechanics™**. *Simulating and Analyzing Mechanical Motion.* 2008. [online]. [Accessed 17th September]. Available from World Wide Web: http://www.mathworks.com/products/simmechanics/description3.htm.

[13] N. Smith and J. Wendlandt, "Creating a Stewart Platform Model Using SimMechanics", *The MathWorks – MATLAB Digest*, Volume 10, Number 5, September 2002.

[14] M.T. Söylemez, "Pole Assignment for Uncertain Systems", *UMIST Control Systems Centre Research Studies,* ISBN 0 86380 246 X, 1999.

[15] xPC Target, "xPC Target User's Guide", The MathWorks, Inc., Natick, MA, 2006.

[16] Pieter J. Mosterman, Sameer Prabhu, and Tom Erkkinen, "An industrial embedded control system design process", *Proceedings of the Inaugural CDEN Design Conference*, July 2004.

[17] Url-3, <http://en.wikipedia.org/wiki/Backlash_(engineering)>, accessed 11.14.2009.

[18] Danaher Motion. *Synchronous Servomotors DBL/DBK Technical Description, Installation, Setup*. Kollmorgen, 2004.

[19] Humusoft. *MF 624 Multifunction I/O Card User' s Manual*, 2006.

[20] Url-3, <http://en.wikipedia.org/wiki/Backlash_(engineering)>, accessed 11.14.2009.

[21] Danaher Motion. *SERVOSTAR 600 Serial Communication Protocol*. Kollmorgen, 2003.

[22] National Instruments. *NI-CAN Hardware and Software Manual*. NI Corp., USA, 2008.

[23] Url-4, <http://www.labbookpages.co.uk/electronics/parallelPort.html>, accessed 11.10.2009.

[24] Simulink 7. *Writing S-Functions*. The MathWorks, Inc., Natick, MA, 2009.

[25] Url-5, < http://www.ni.com/lwcvi/ >, accessed 03.10.2009.

[26] Danaher Motion. *Driver.exe Software for SERVOSTAR400/600*. Kollmorgen, 2003.

[27] Micro-Radian Instruments. *Autocollimator Operating and Technical Manual*. Micro-Radian Instruments Inc., USA, 2008.

[28] Brüel & Kjær. *PULSE Multi-analyzer System Type 3560 B/C/D/E Installation Manual*. Brüel & Kjær Inc., Denmark, 2005.

APPENDICES

APPENDIX A.1

	Data	Symbol [Unit]	DBL3 N00065	DBL3 H00065	DBL3 N00130	DBL3 H00130	DBL3 M00190	DBL3 H00250	DBL3 N00300
Electrical data									
	Standstill torque	**M_0 [Nm]**	**0,65**	**0,65**	**1,3**	**1,3**	**1,9**	**2,5**	**3,0**
	Standstill current	**I_{0rms} [A]**	**0,67**	**1,08**	**1,0**	**1,75**	**1,5**	**3,0**	**2,1**
	Mains voltage	U_N [VAC]	230-480						
U_N = 230V	Rated speed	n_n [min^{-1}]	—	3000	—	3000	—	3000	—
	Rated torque	M_n [Nm]	—	0,6	—	1,2	—	2,2	—
	Rated current	I_n [A]	—	1,05	—	1,6	—	2,7	—
	Rated power	P_n [kW]	—	0,19	—	0,38	—	0,69	—
U_N = 400V	**Rated speed**	**n_n [min^{-1}]**	**3000**	**6000**	**3000**	**6000**	**3000**	**6000**	**3000**
	Rated torque	M_n [Nm]	0,60	0,48	1,20	1,1	1,6	1,80	2,6
	Rated current	I_n [A]	0,65	0,95	0,95	1,5	1,32	2,40	1,90
	Rated power	P_n [kW]	0,19	0,30	0,38	0,69	0,50	1,13	0,82
U_N = 480V	Rated speed	n_n [min^{-1}]	3600	—	3600	—	3600	—	3600
	Rated torque	M_n [Nm]	0,58	—	1,15	—	1,54	—	2,5
	Rated current	I_n [A]	0,59	—	0,90	—	1,21	—	1,73
	Rated power	P_n [kW]	0,22	—	0,43	—	0,58	—	0,94
	Peak current	I_{0max} [A]	3,0	5,0	4,5	7,5	6,9	13,9	9,5
	Torque constant	$K_{T rms}$ [Nm/A]	0,98	0,60	1,28	0,74	1,27	0,83	1,46
	Voltage constant	$K_{E rms}$ [mVmin]	59	36,5	77,5	45	77	50	88
	Winding resistance Ph-Ph	R_{20} [Ω]	79	30,3	35,5	13	21,3	5,1	11,5
	Winding inductance Ph-Ph	L [mH]	82,8	31	61	22	40	11	25
Mechanical data									
	Rotor moment of inertia	J [kgcm²]	0,5		0,8		1,0	1,4	1,7
	Static friction torque	M_R [Nm]	0,02		0,02		0,03	0,05	0,05
	Thermal time constant	t_{TH} [min]	25		30		32	32	35
	Weight standard	G [kg]	1,9		2,3		2,5	3,3	4
	Radial load permitted at shaft end @ 3000 min^{-1}	F_R [N]	350						
	Axial load permitted at shaft end @ 3000 min^{-1}	F_A [N]	110						
	Motor number		00299R	00276R	00258R	00275R	00263R	00420R	00252R

Brake data

Data	Symbol [Unit]	Value
Holding torque	M_{BR} [Nm]	2,5
Operating voltage	U_{BR} [VDC]	24 +15 / -0 %
electrical power	P_{BR} [W]	12
Moment of inertia	J_{BR} [kgcm²]	0,38
Release delay time	t_{BRH} [ms]	10-15
Application delay time	t_{BRL} [ms]	10-15
Weight of the brake	G_{BR} [kg]	0,4

Connections and leads

Data	DBL3 N00065	DBL3 H00065	DBL3 N00130	DBL3 H00130	DBL3 M00190	DBL3 H00250	DBL3 N00300
Power connection	4 + 4 poles, round, angular						
Motor cable, shielded	4 x 1						
Motor cable with control leads, shielded	4 x 1 + 2 x 0,75						
Resolver connection	12 poles, round, angular						
Resolver cable, shielded	4 x 2 x 0,25mm²						
Encoder connection (Option)	17 poles, round, angular						
Encoder cable, shielded	7 x 2 x 0,25mm²						

Figure A.1 : Technical data of DBL3N00300 type synchronous servomotor.

		SERVOSTAR						
Rated data	**DIM**	**601**	**603**	**606**	**610**	**610-30**	**614**	**620**
Rated supply voltage (grounded system)	V~	3 x 230V-10% ... 480V+10%, 50 Hz						
	V~	3 x 208V-10% ... 480V+10%, 60 Hz						
Rated installed load for S1 operation	kVA	1	2	4	7	7	10	14
Rated DC bus link voltage	V=	260 - 675						
Rated output current (rms value, ± 3%)	A_{rms}	1.5	3	6	10	10	14	20
Peak output current (max. ca. 5s, ± 3%)	A_{rms}	3	6	12	20	30 (2s)	28	40
Clock frequency of the output stage	kHz	8 (16 with VDCmax=400V)						
Technical data for regen circuit	—	⇨ p.22						
Overvoltage protection threshold	V	450...900						
Max. load inductance	mH	150	75	40	25	24	15	12
Min. load inductance	mH	25	12	7.5	4	4	2.5	2
Form factor of the output current (at rated data and min. load inductance)	—	1.01						
Bandwidth of subordinate current controller	kHz	> 1.2						
Residual voltage drop at rated current	V	5						
Quiescent dissipation, output stage disabled	W	15						
Dissipation at rated current (incl. power supply losses, without regen dissipation)	W	30	40	60	90	90	160	200
Inputs								
Setpoint 1/2, resolution 14bit/12bit	V	±10						
Common-mode voltage max.	V	±10						
Input resistance to AGND	kΩ	20						
Digital inputs	V	according to IEC 61131						
Digital outputs, open collector	V	according to IEC 61131						
BTB/RTO output, relay contacts	V	DC max. 30, AC max. 42						
	mA	500						
Aux. power supply, electrically isolated	V	24 (-0% +15%)						
without brake	A	1 (max. 16)						
Aux. power supply, electrically isolated	V	24 (-0% +15%)						
with brake (consider voltage loss!)	A	3 (max. 16)						
Min./max. output current, brake	A	0,15 / 2						
Connections								
Control signals	—	Combicon 5.08 / 18 pole , 2,5mm²						
Power signals	—	Power Combicon 7.62 / 4x4 + 1x6-pole, 4mm²						
Resolver input	—	SubD 9pole (socket)						
Sine-cosine encoder input	—	SubD 15pole (socket)						
PC-interface, CAN	—	SubD 9pole (plug)						
Encoder simulation, ROD (EEO) / SSI	—	SubD 9pole (plug)						
Mechanical								
Weight	kg	4					5	7.5
Height without connectors	mm	275						
Width	mm	70					100	120
Depth without connectors	mm	265						

Figure A.2 : Technical data of SERVOSTAR amplifier.

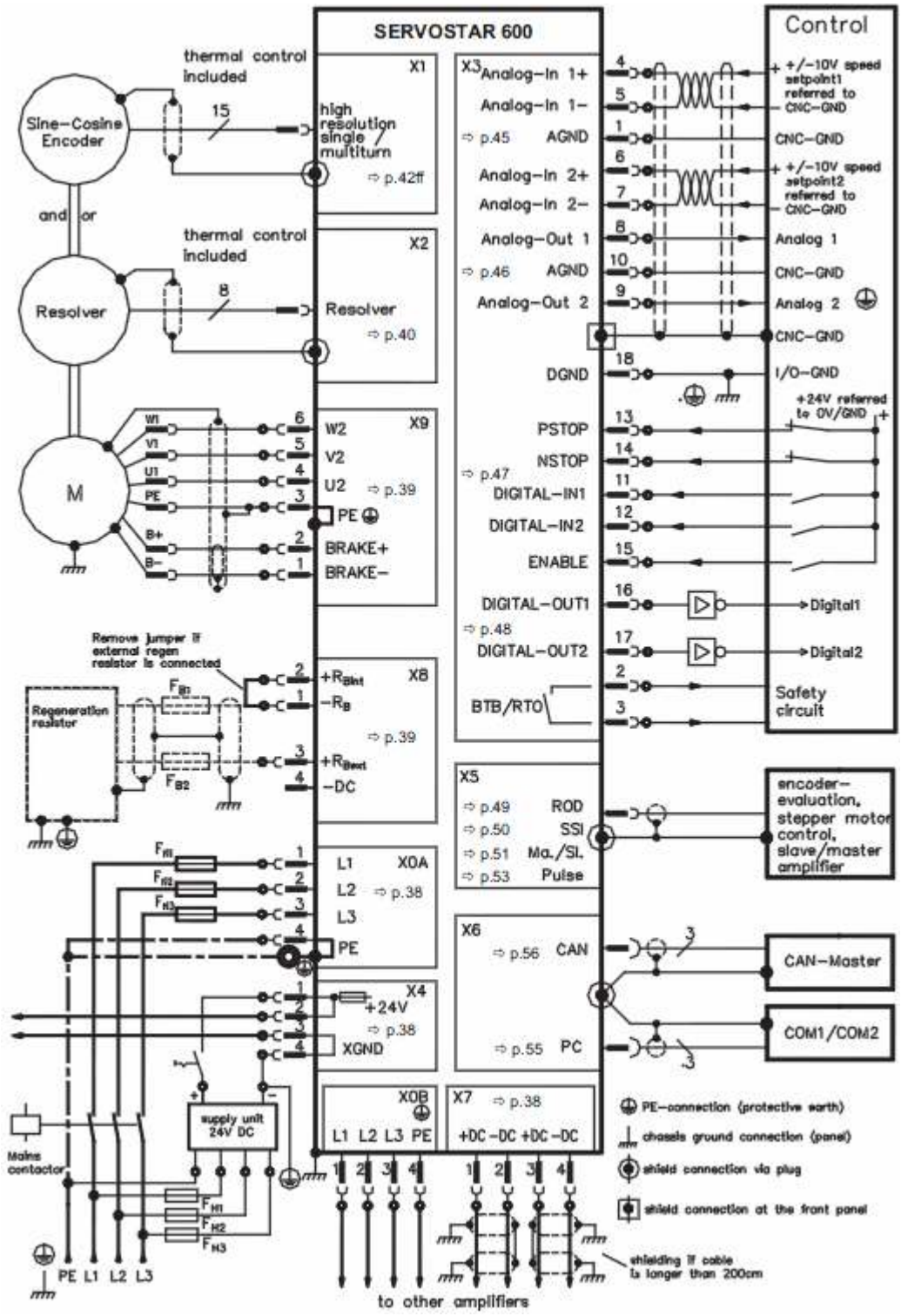

Figure A.3 : Connection diagram of SERVOSTAR amplifier.

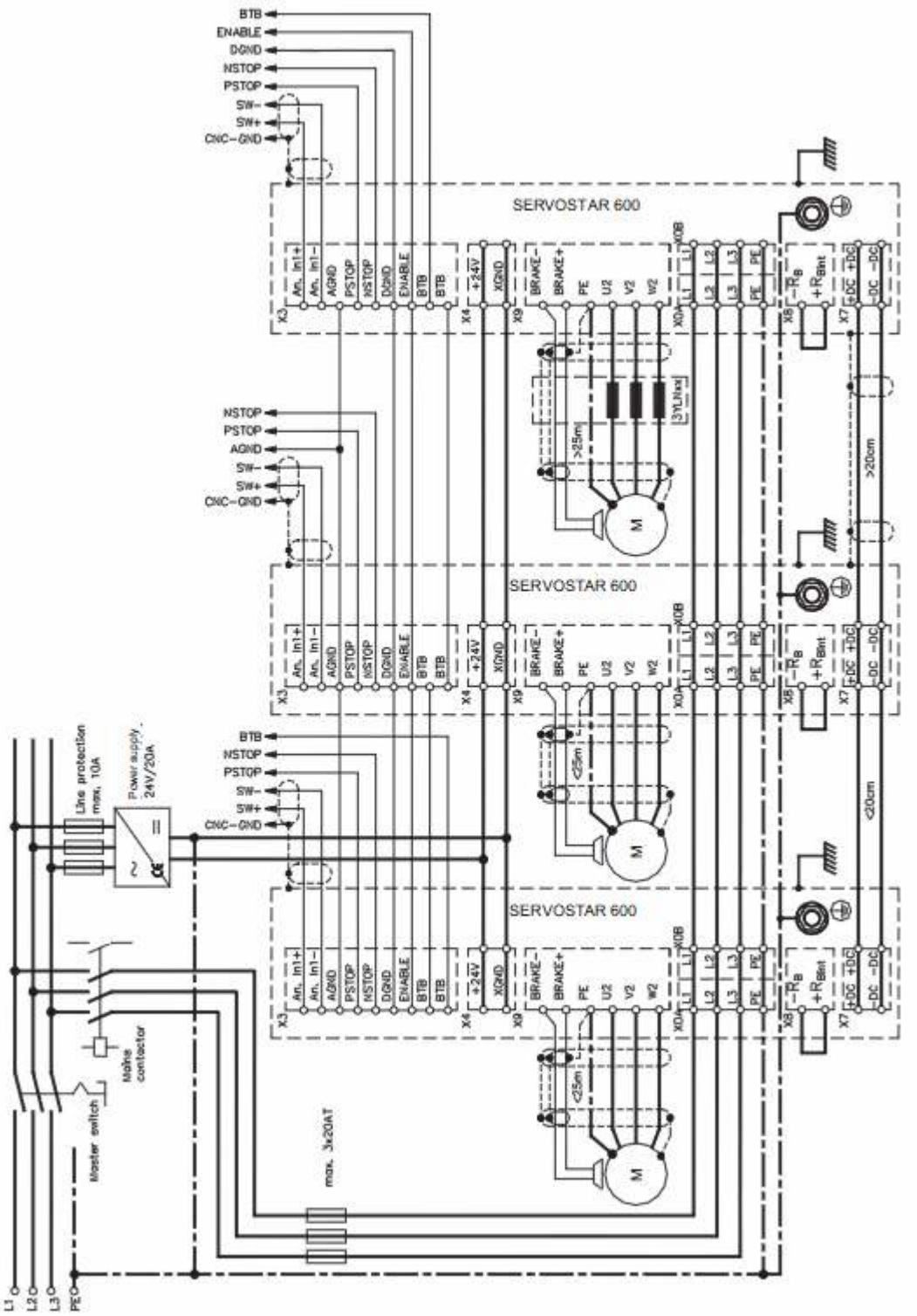

Figure A.4 : Example of connections for multi-axis system.

APPENDIX A.2

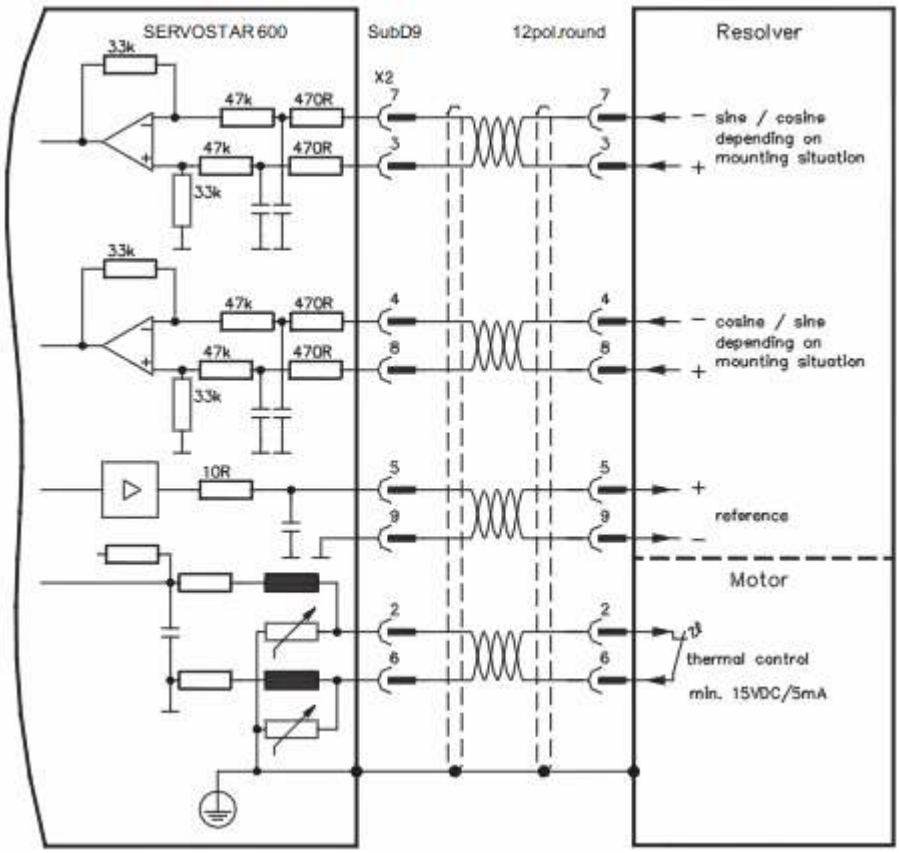

Figure A.5 : Amplifier-resolver connection.

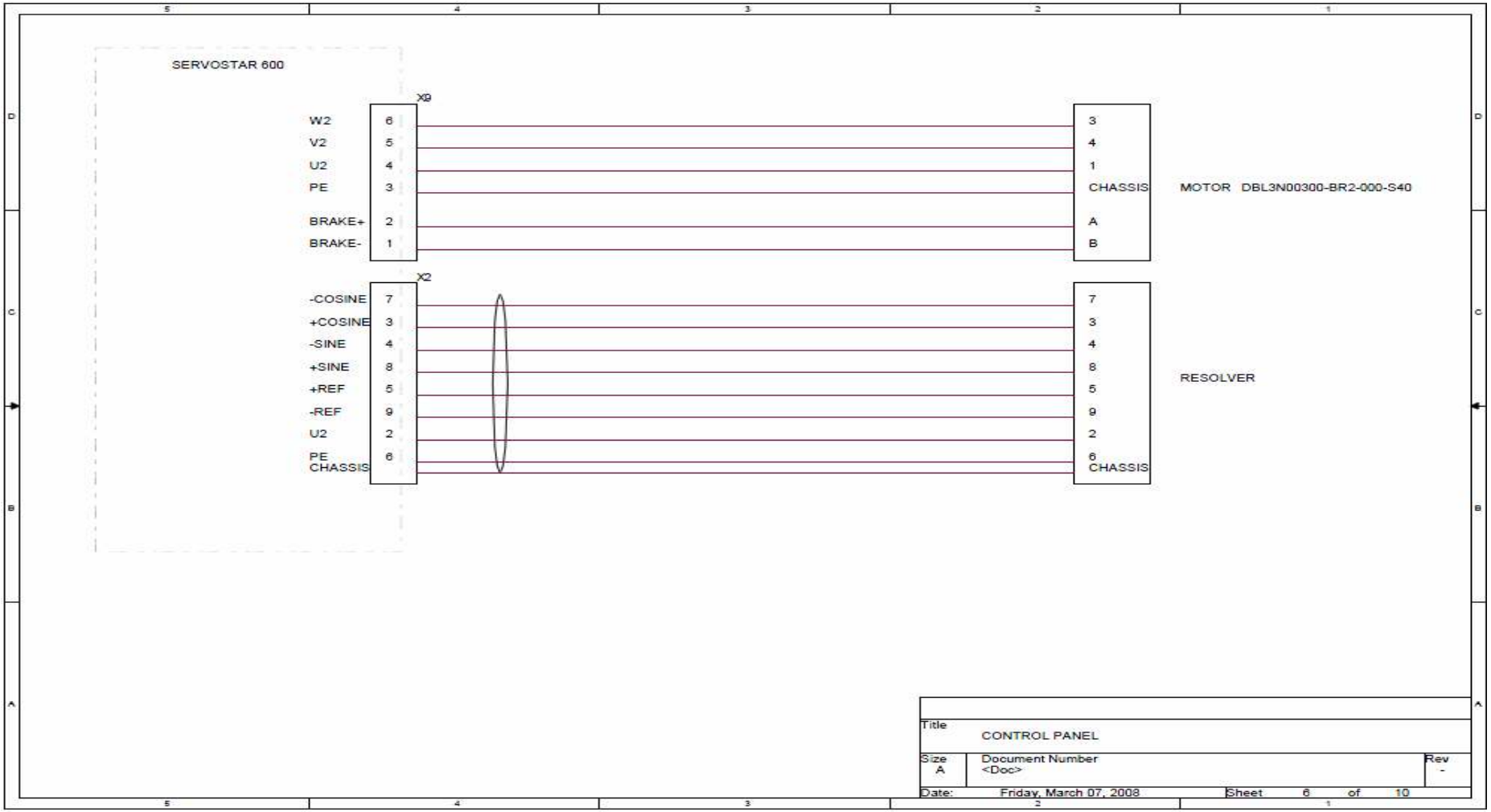

Figure A.6 : Amplifier-motor&resolver wiring schematic.

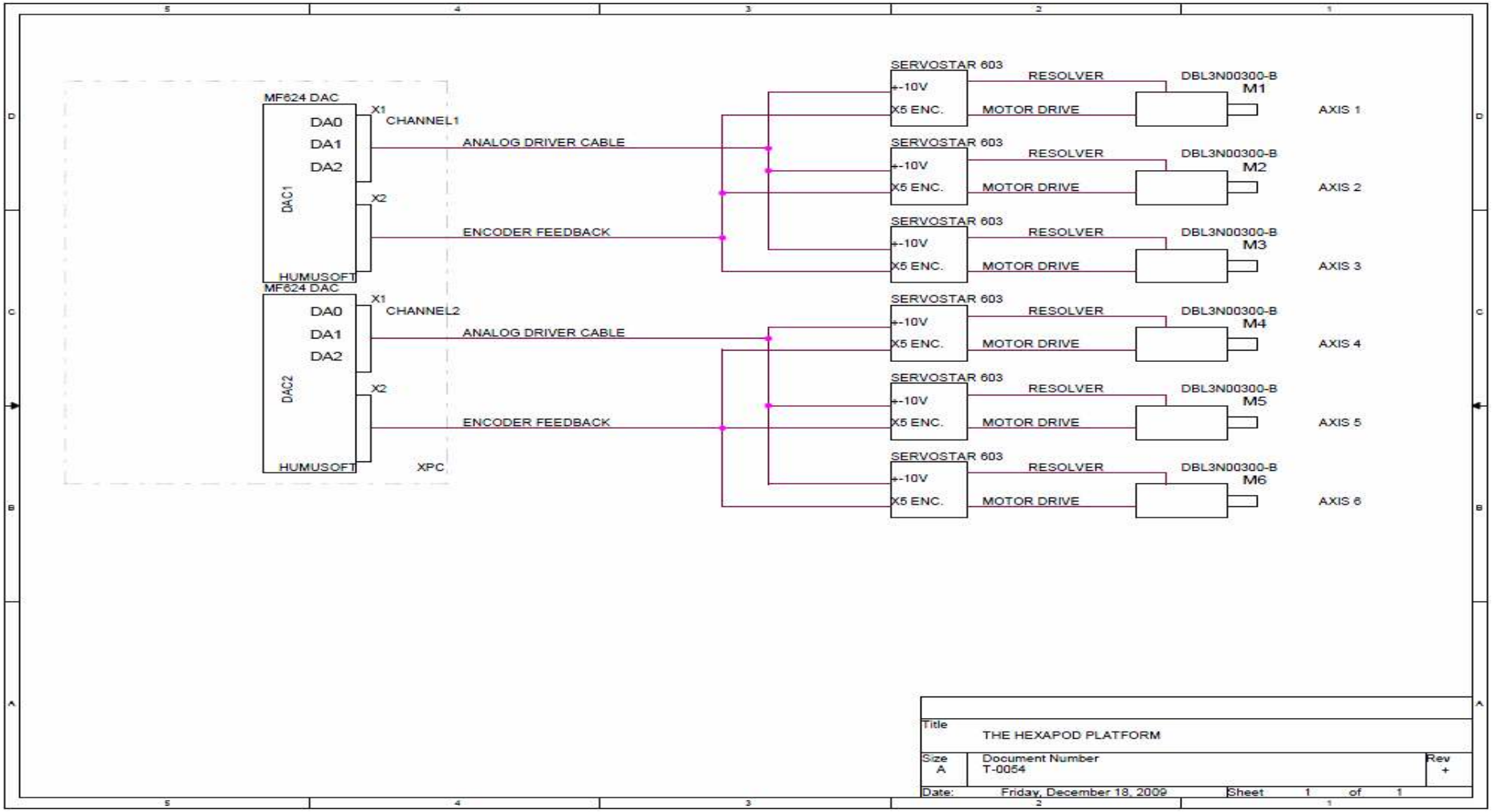

Figure A.7 : Humusoft MF 624- motor&resolver wiring schematic.

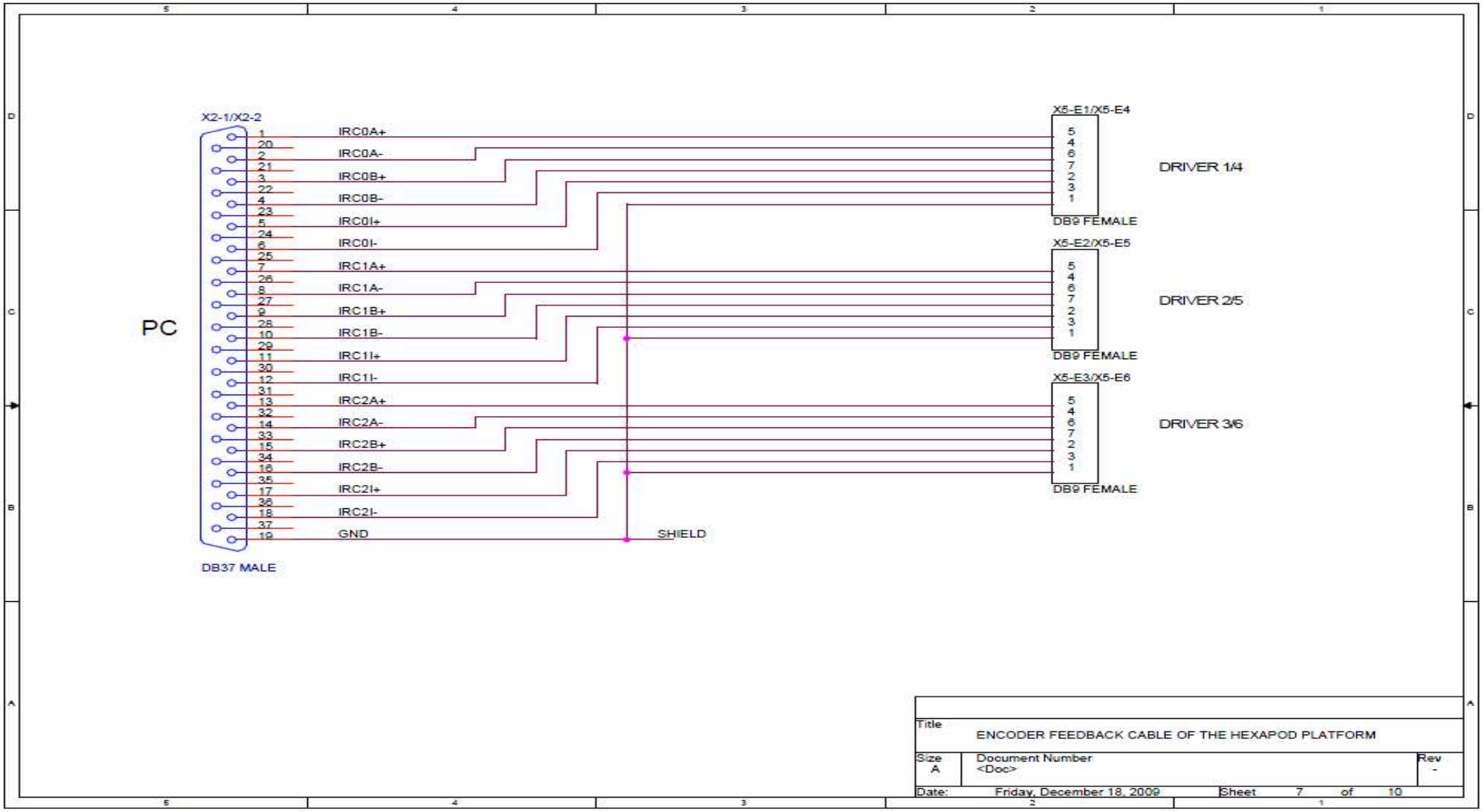

Figure A.8 : Encoder-Humusoft MF 624 cable wiring schematic

APPENDIX A.3

The MF 624 multifunction I/O card is designed for the need of connecting PC compatible computers to real world signals. The MF 624 contains 8 channel fast 14 bit A/D converter with simultaneous sample/hold circuit, 8 independent 14 bit D/A converters, 8 bit digital input port and 8 bit digital output port, 4 quadrature encoder inputs with single-ended or differential interface and 5 timers/counters. The card is designed for standard data acquisition and control applications and optimized for use with Real Time Toolbox for Simulink®. MF 624 features fully 32-bit architecture for fast throughput.

A/D Converter

Resolution: 14 bits

Number of channels: 8 single ended

Sample/hold circuit: simultaneous sampling of all channels

Conversion time: 1.6 μs single channel

1.9 μs 2 channels

2.5 μs 4 channels

3.7 μs 8 channels

FIFO: 8 entries/one conversion cycle

Input ranges: ±10V

Input protection: ±18V

Input impedance: > 10_{10} Ohm

D/A Converter

Resolution: 14 bit

Number of channels: 8

Settling time: max. 31 μs (full scale swing, 1/2 LSB)

Slew Rate: 10 V/µs

Output current: min. ±10 mA

Short circuit current: ±15 mA

DC output impedance: max. 0.5 Ohm

Load capacitance: max. 50 pF

Differential nonlinearity: ±1 LSB

Digital Inputs

Number of bits: 8

Input signal levels: TTL

Logic 0: 0.8 V max.

Logic 1: 2.0 V min.

Quadrature Encoder Inputs

Number of axes: 4 independent

Resolution: 32 bits

Counter modes: quadrature X4 or up/down counter

Index input: programmable

Inputs: differential with Schmitt triggers

Input noise filter: digital, programmable (0.3 µs)

Input frequency: max. 2.5 MHz

Counters/Timers

Counter chip: custom

Number of channels: 5, 4 of them available on I/O connector, one used for

A/D triggering and interrupt

Resolution: 32 bits

Clock frequency: 50 MHZ

Counter modes: up, down, binary

Triggering: software, external

Clock source: internal, prescalers, external

Inputs: TTL, Schmitt triggers

Outputs: TTL

APPENDIX B.1

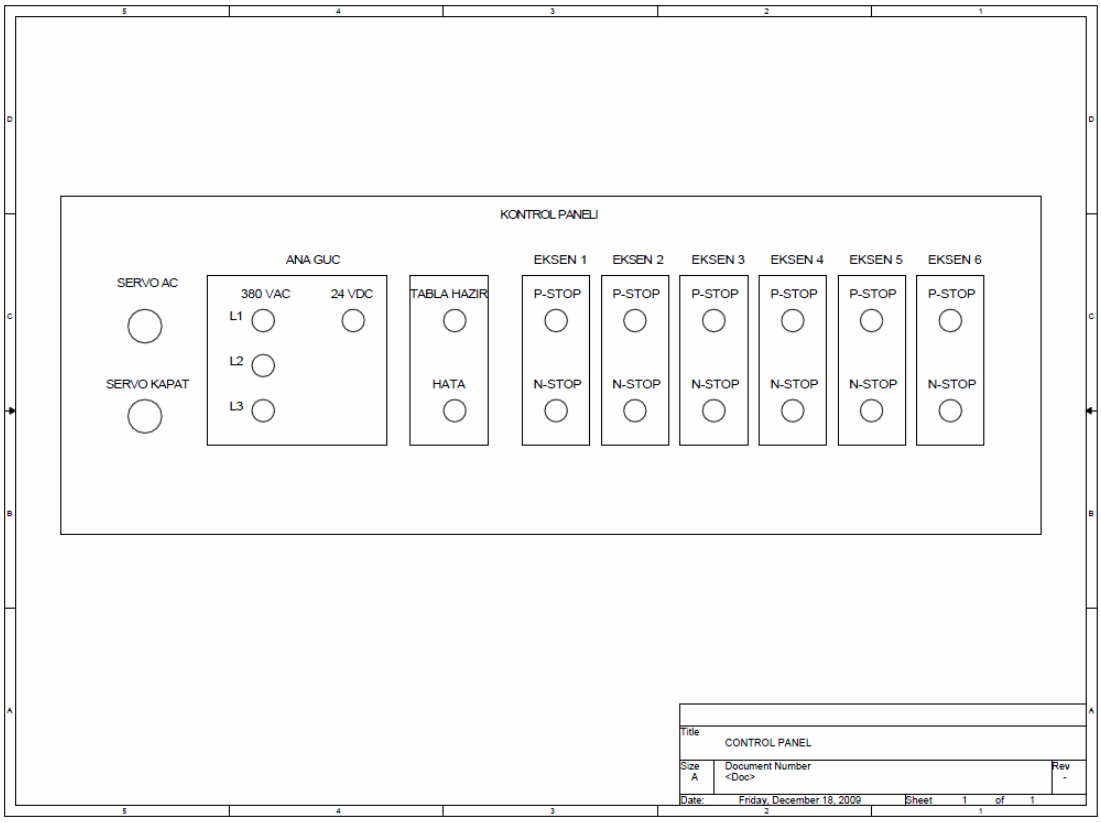

Figure B.1 : Control panel front view.

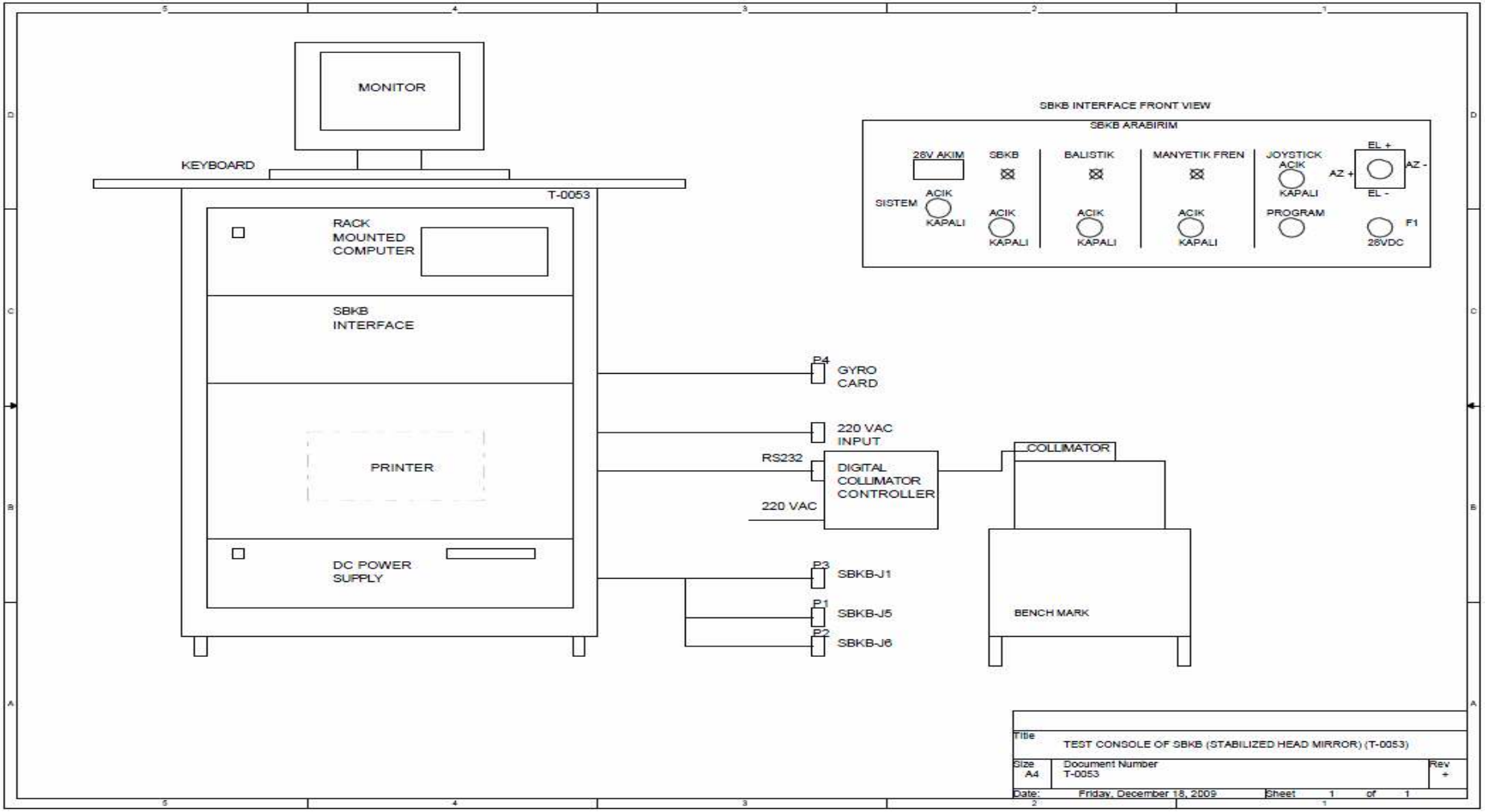

Figure B.2 : Control panel front view.

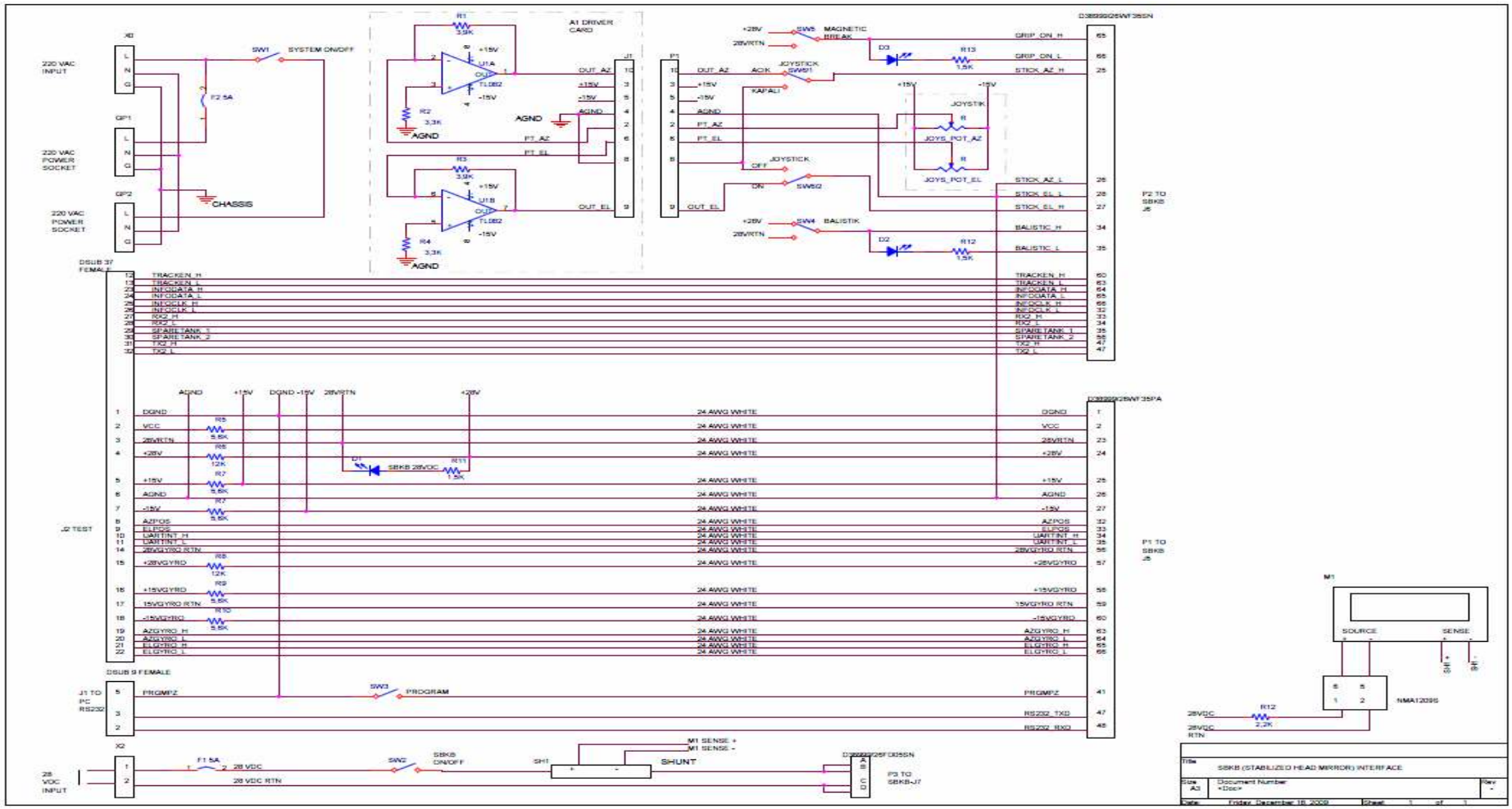

Figure B.3 : Electrical interface of the stabilized head mirror.

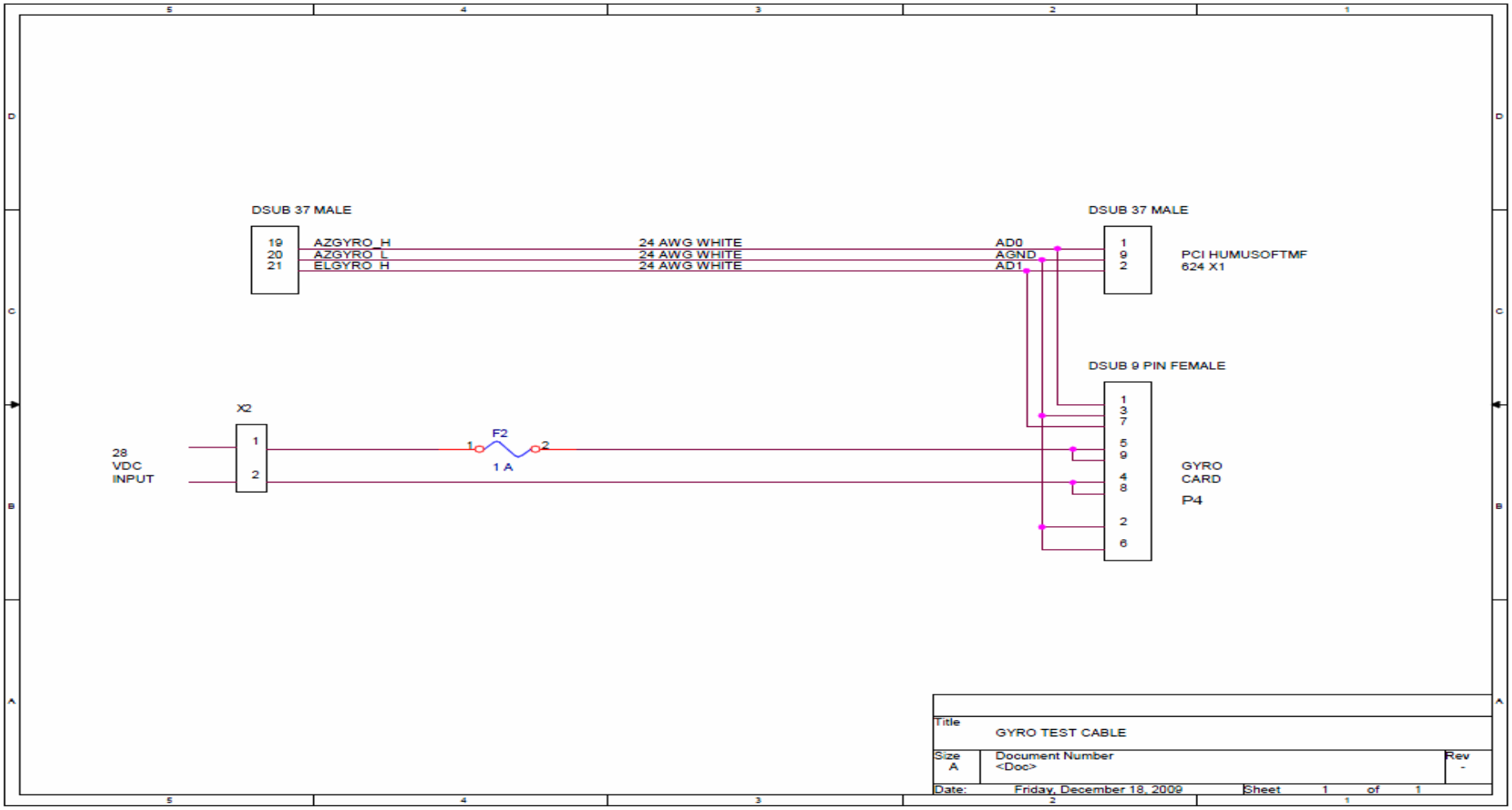

Figure B.4 : The stabilized head mirror' s gyro test cable.

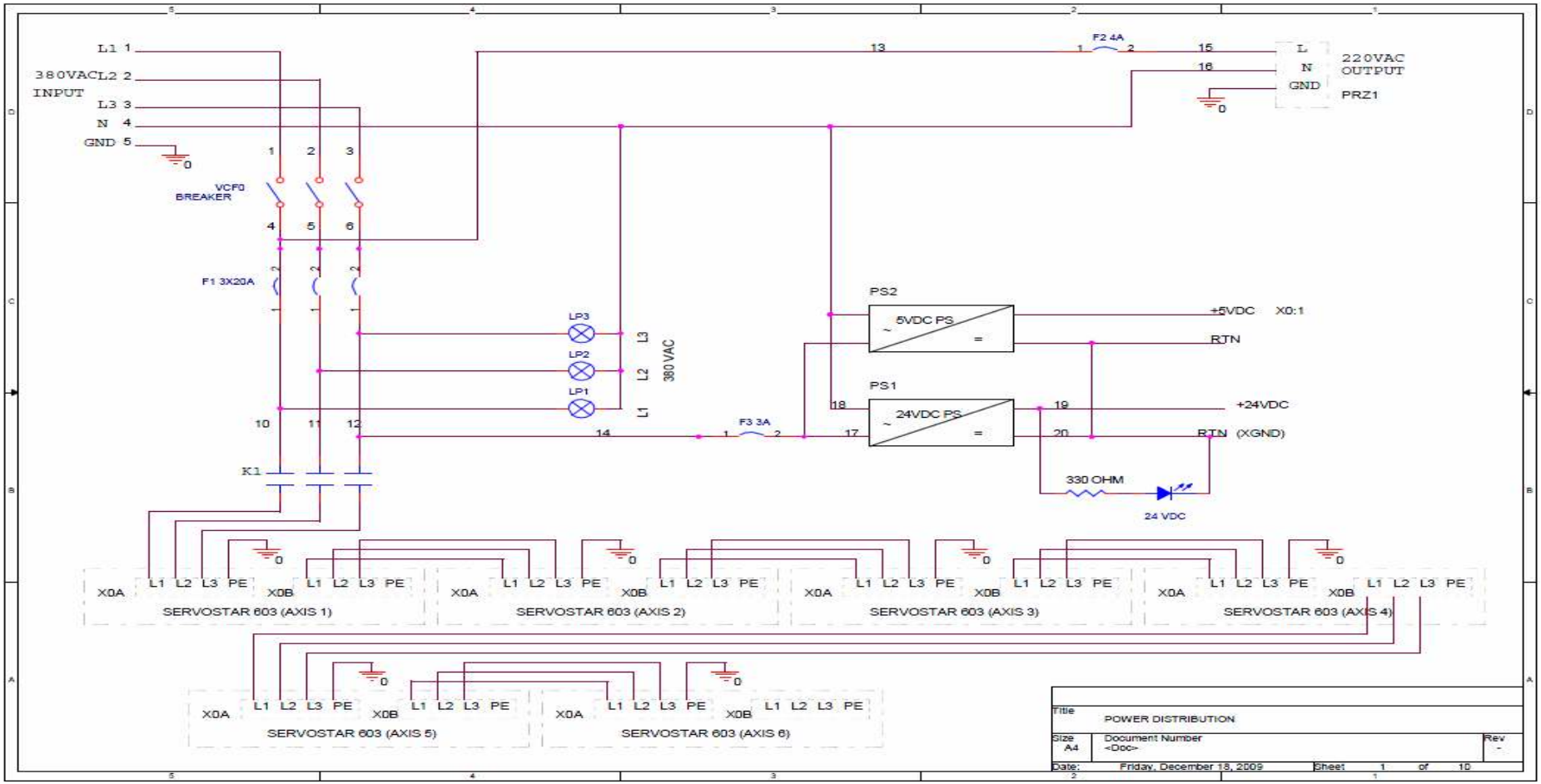

Figure B.5 : Power distribution.

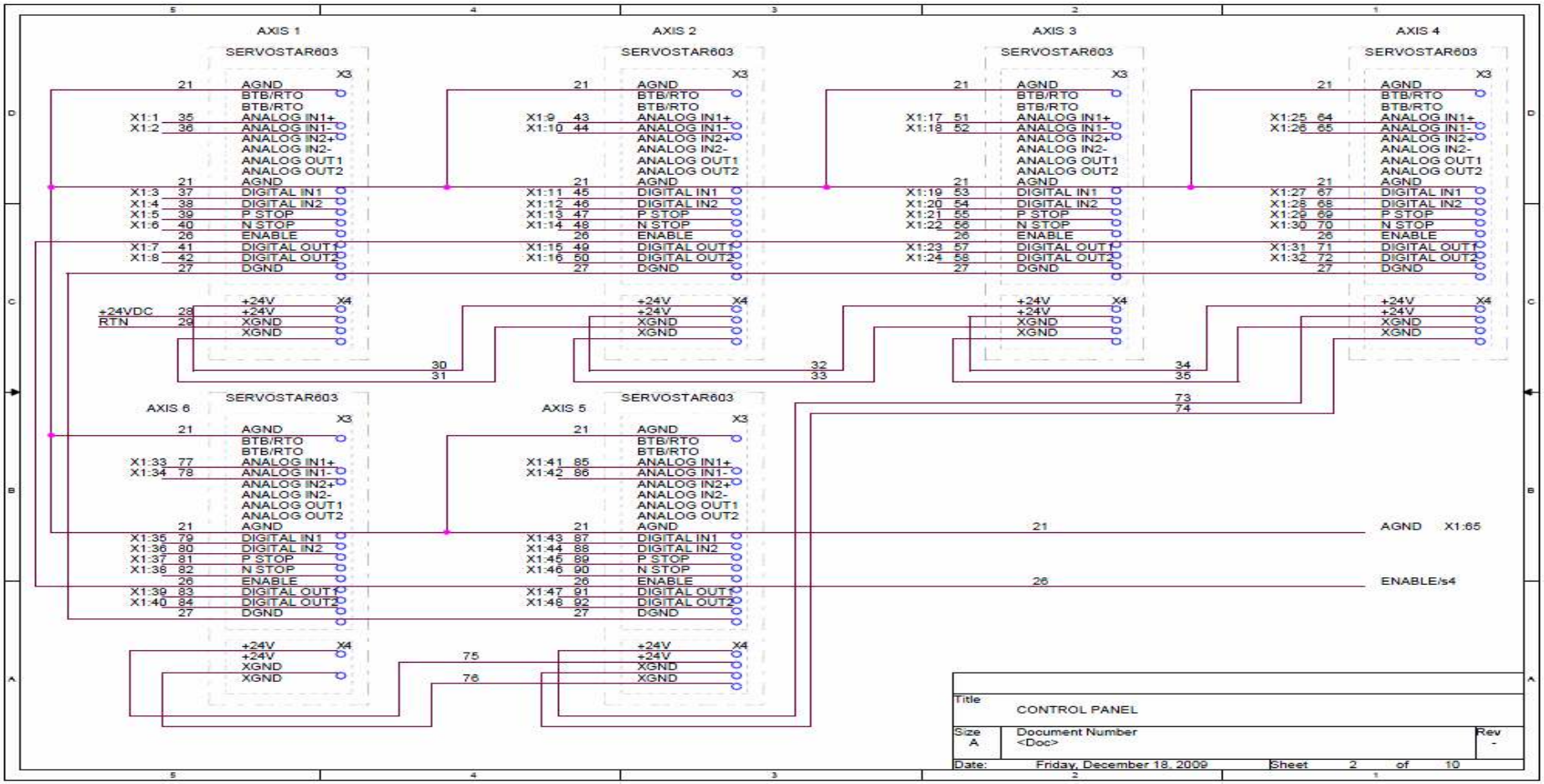

Figure B.6 : First part of control panel.

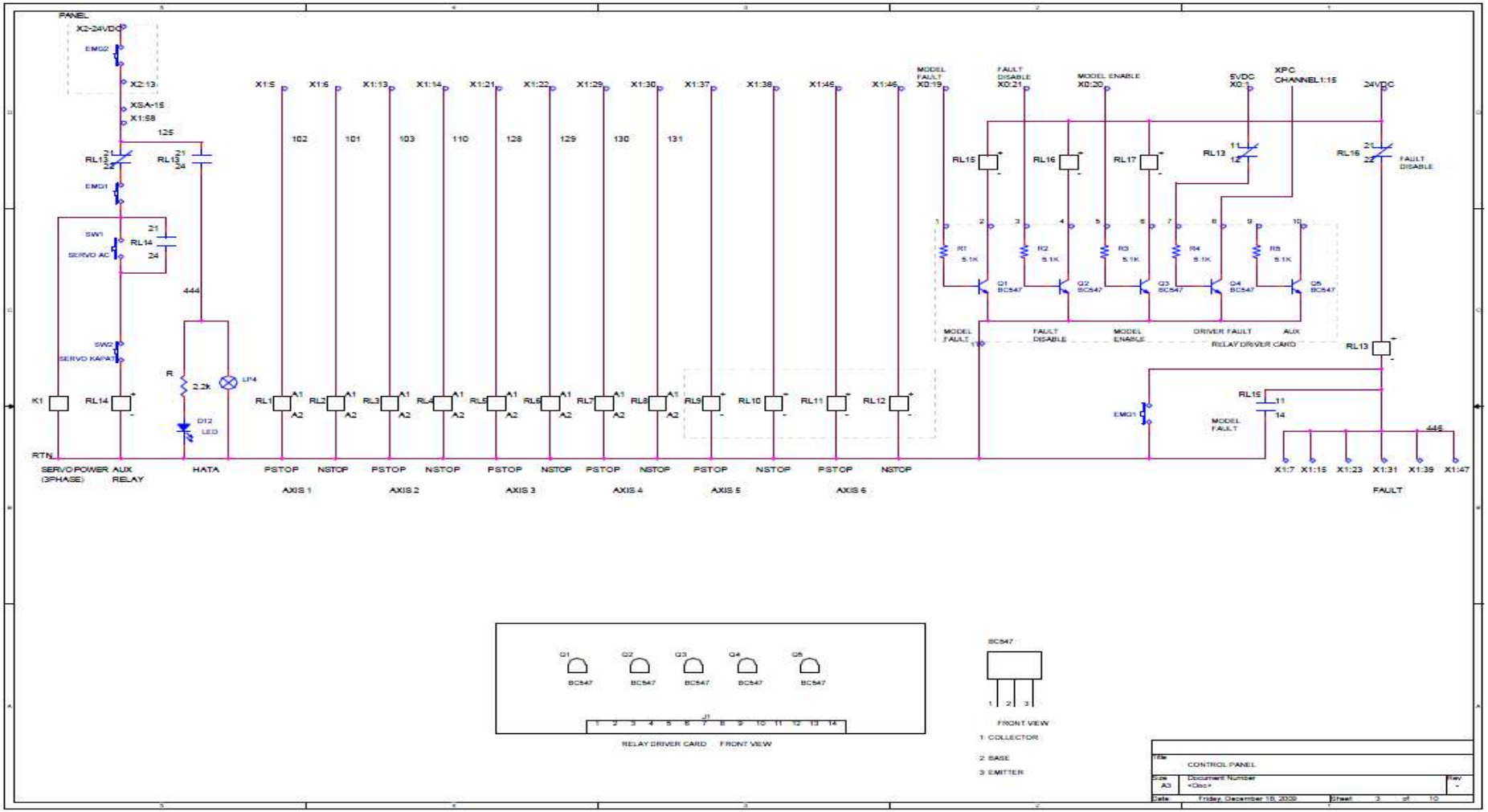

Figure B.7 : Second part of control panel.

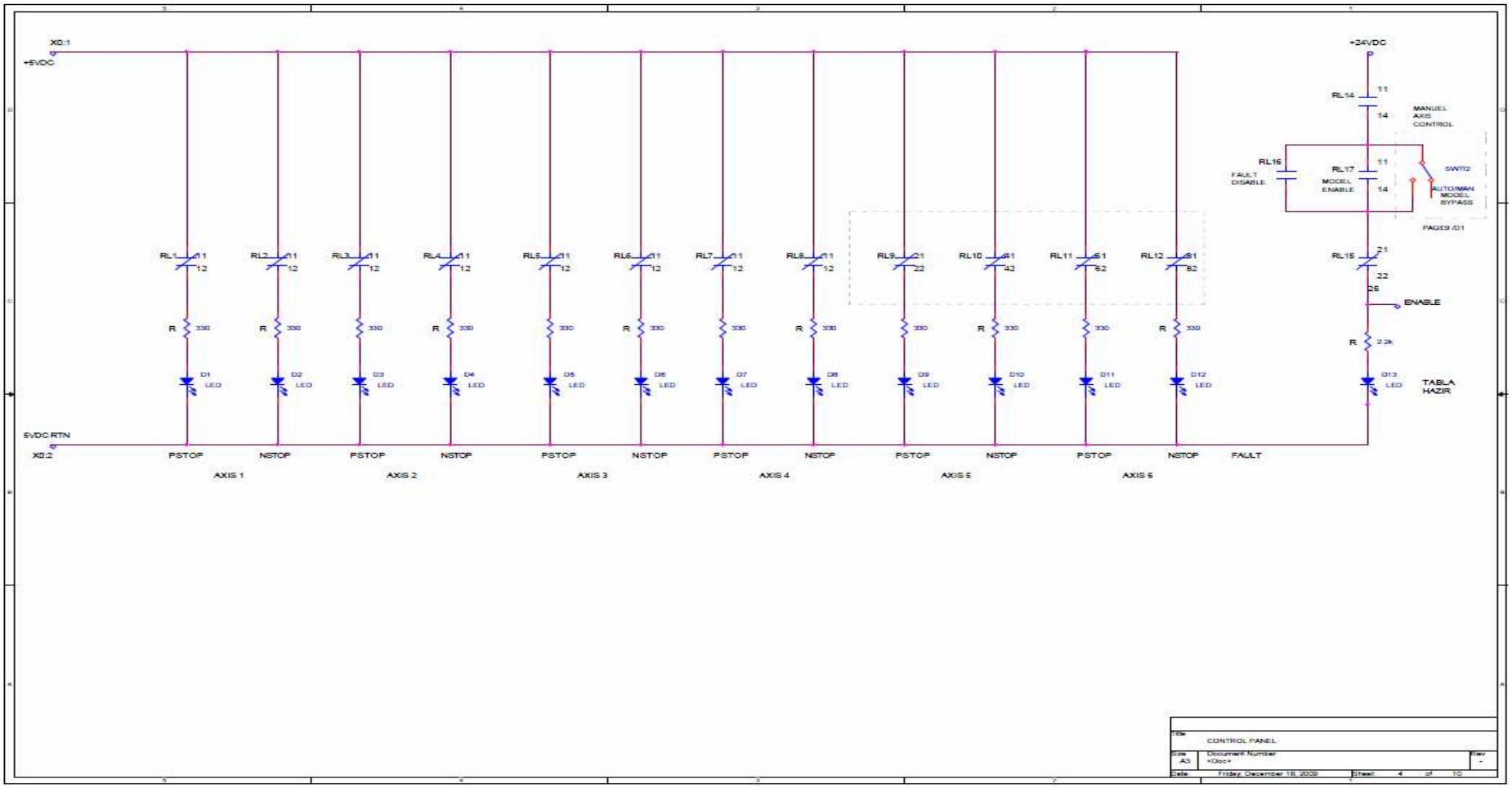

Figure B.8 : Third part of control panel.

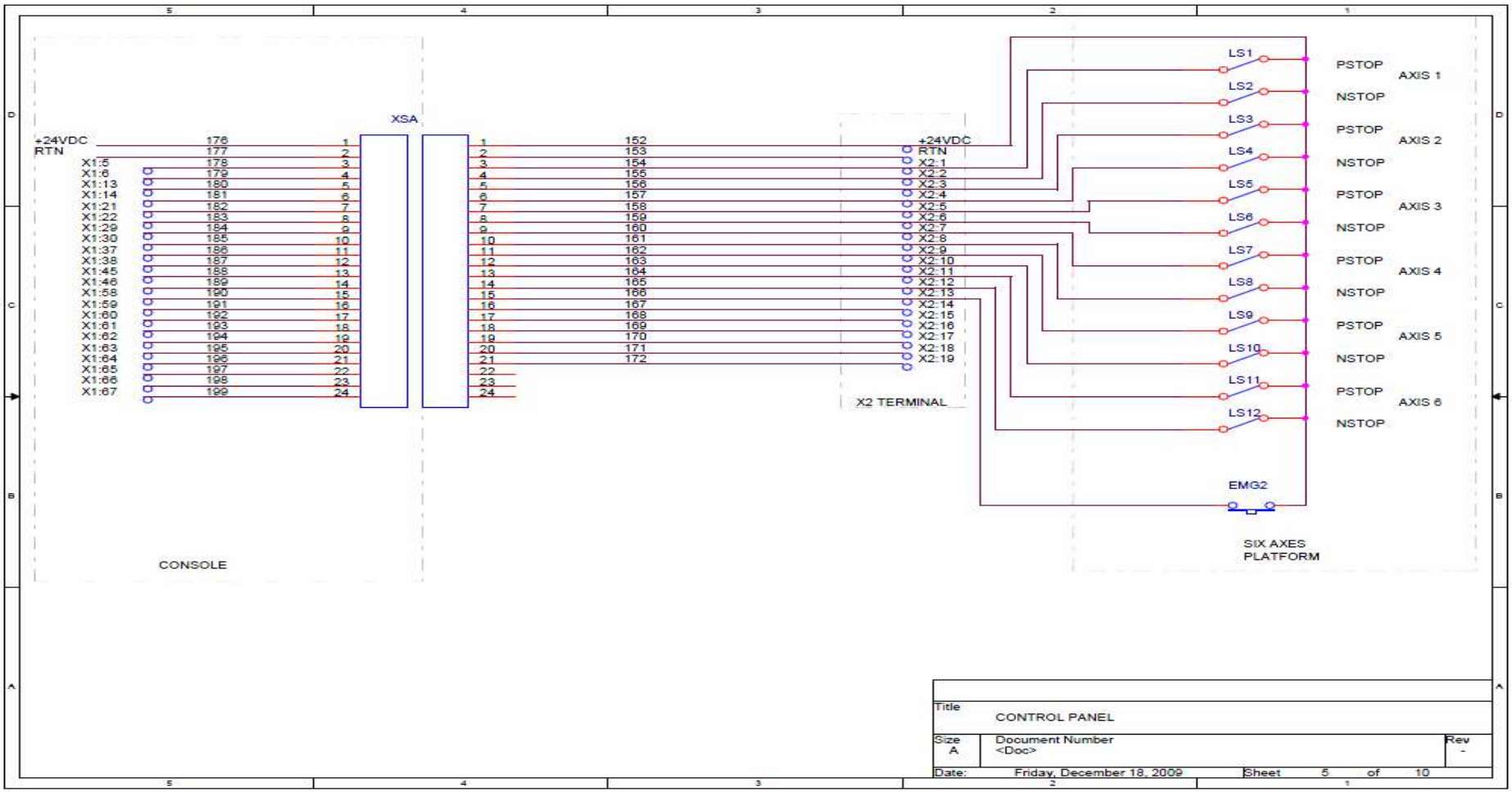

Figure B.9 : Fourth part of control panel.

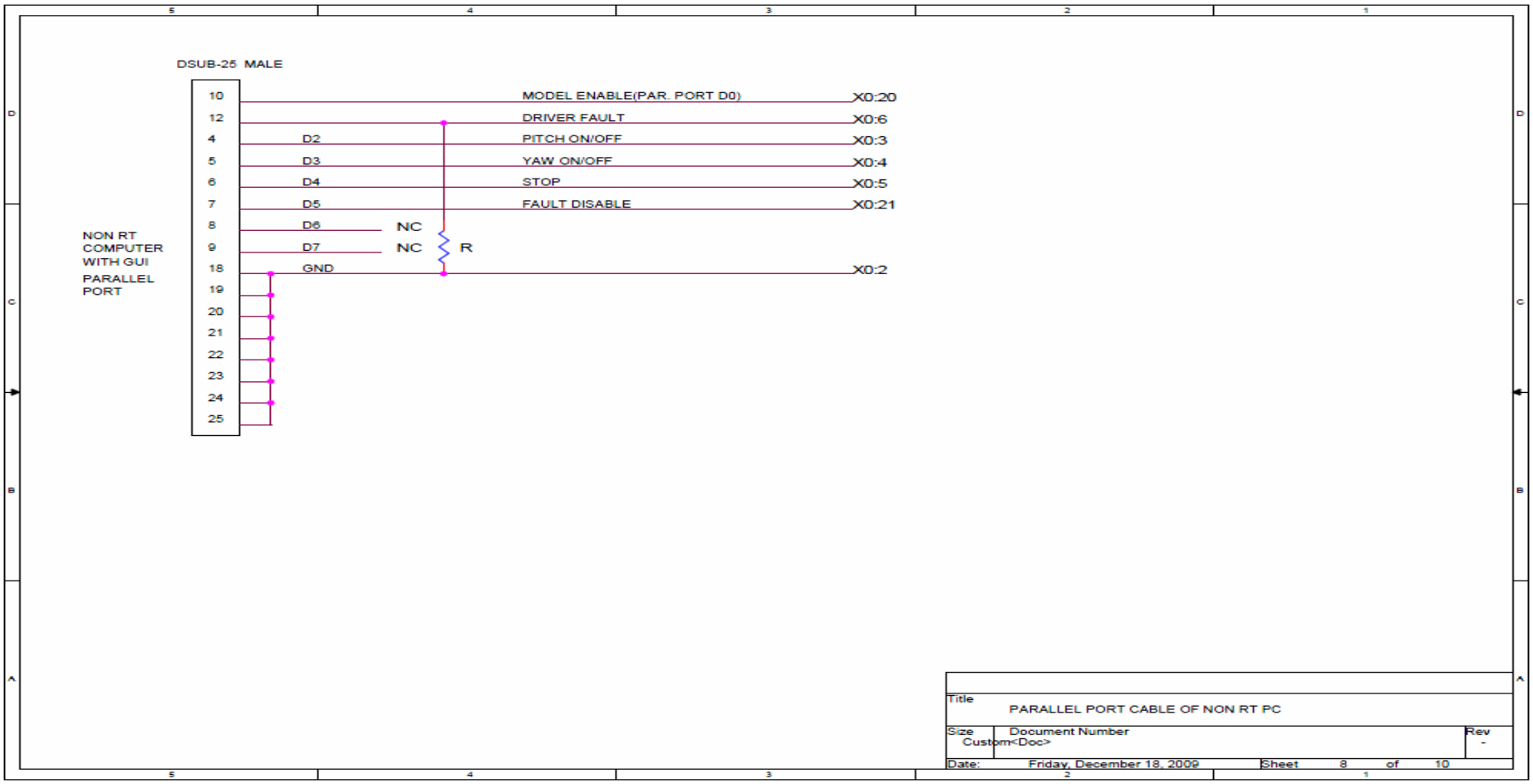

Figure B.10 : Parallel port-MF 624 DIO connection.

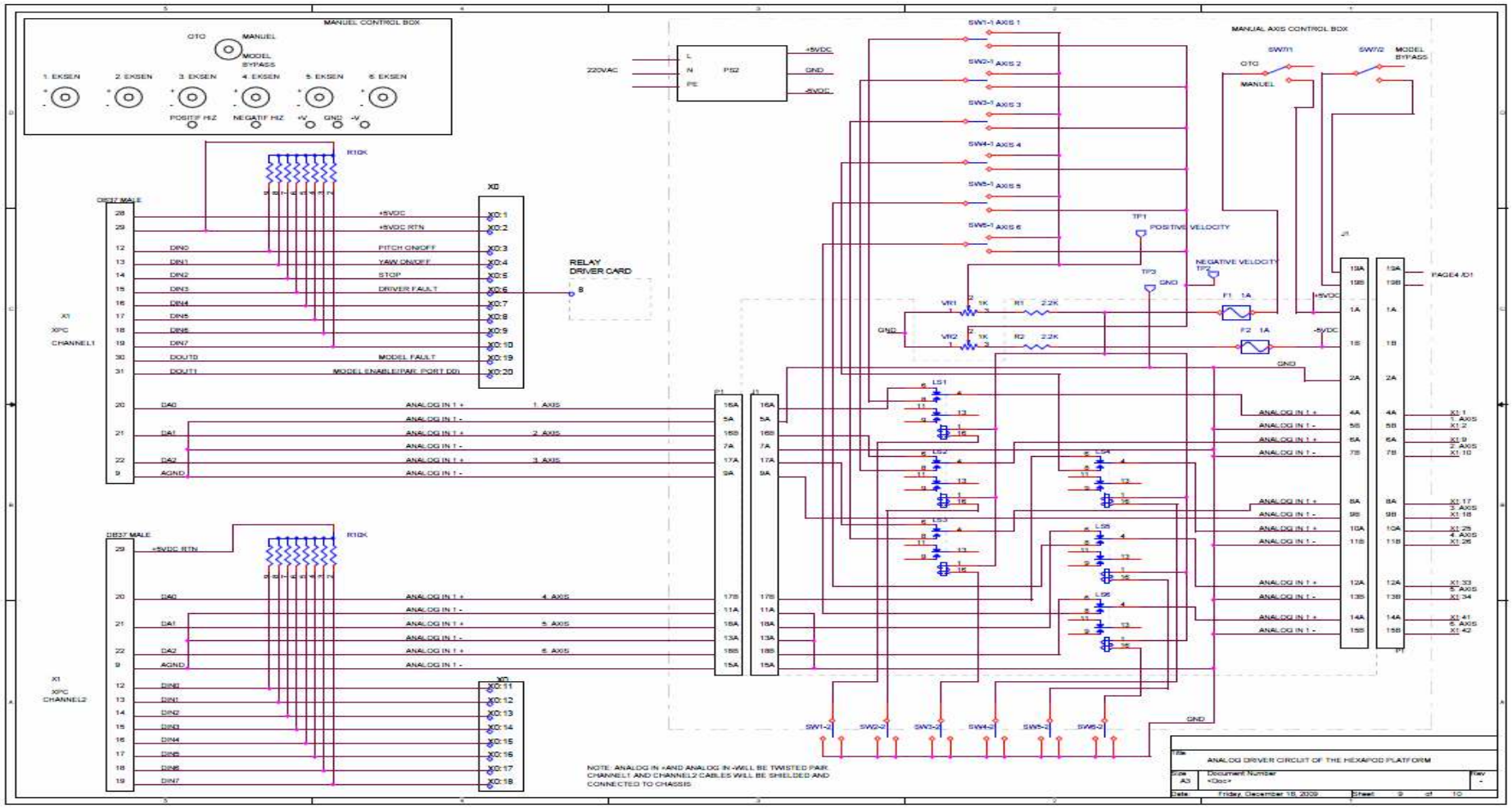

Figure B.11 : Analog driver circuit of the Hexapod platform.

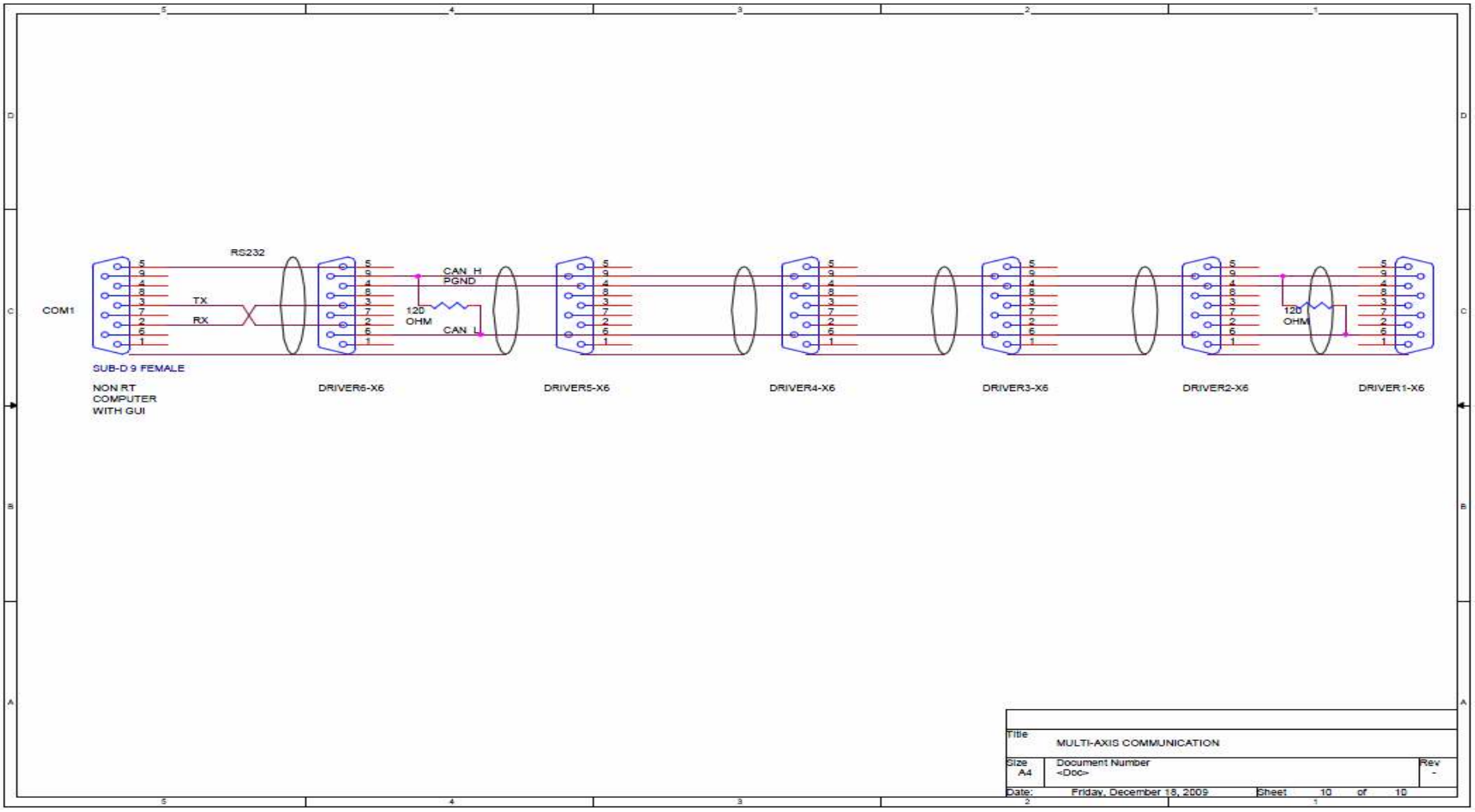

Figure B.12 : RS232-CAN bus connection diagram for multi-axis communication.

APPENDIX C.1

All variables are measured in meters.

umm_arm_length (upper or lower leg length) = 0,4610

umm_body_pts (position vectors of upper legs' connection points on top plate)=

$$\begin{bmatrix} x_{upperlegconnectionpts_i} \\ y_{upperlegconnectionpts_i} \\ z_{upperlegconnectionpts_i} \end{bmatrix} = \begin{bmatrix} 0.2832 & 0.2832 & 0.1444 & -0.4276 & -0.4276 & 0.1444 \\ -0.3303 & 0.3303 & 0.4104 & 0.0801 & -0.0801 & -0.4104 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

umm_leg_length (distance between lower and upper legs' connection points on plates that corresponds 120 degrees, nominal angle, between lower and upper legs at home, initial, position)=

$$\left[nomıomınalLength_i\right] = \begin{bmatrix} 0.7985 & 0.7985 & 0.7985 & 0.7985 & 0.7985 & 0.7985 \end{bmatrix}$$

umm_pos_base (position vectors of lower legs' connection points on base plate)=

$$\begin{bmatrix} x_{lowerlegconnectionpts_i} \\ y_{lowerlegconnectionpts_i} \\ z_{lowerlegconnectionpts_i} \end{bmatrix} = \begin{bmatrix} 0.6518 & 0.6518 & -0.2586 & -0.3932 & -0.3932 & -0.2586 \\ -0.0777 & 0.0777 & 0.6034 & 0.5256 & -0.5256 & -0.6034 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

APPENDIX C.2

poleassign.m

```
%   POLEASSSIGN  'Pole assignment function for unit feedback
systems.'
%
%   Assignment : Numbers of Free Poles of the Desired Controller of
%   the Closed Loop System can be assigned at desired points on
complex
%   plane.
%
%
%                 Controller                       Plant
%     R--->O--->[C_Num/C_Denum]-------------->[P_Num/P_Denum]----+--
-> Y
%         -|                                                     |
%          +-----------------------------------------------------+
%
%    Help:
%
%    Sample,
%
%    Define Controller Type: (PID, 1.Order, ...etc...):
%    PID  -> (Kd*s^2 + Kp*s + Ki)/s
%    Define Symbolic Parameters of the Controller:
%    syms  Kd Kp Ki;
%    Define Numerator and Denumerator of the Controller:
%    C_Num = [Kd Kp Ki];
%    C_Denum = [1 0];
%    Define Plant Transfer Function:
%    s = tf('s');
%    Gs = 329.511/(s^2+1.12*s);
%    P_Num = 329.511;  -> Plant Numerator
%    P_Denum = [1 1.12 0];   -> Plant Denumerator
%    Define Closed Loop Pole Locations:
%    CL_Poles = [-16+16.0062*j -16-16.0062*j -80];
%    Call poleassign function:
%    Gc=poleassign(P_Num, P_Denum, CL_Poles, C_Num, C_Denum)
%
%    Transfer function:
%    0.3365 s^2 + 9.324 s + 124.4
%    ----------------------------
%                 s
%

function [Gc]=poleassign(P_Num,P_Denum,CL_Poles,C_Num,C_Denum)
c=[C_Num C_Denum];
f=c;
kp1=symbolicmultiply(P_Denum,C_Denum);
kp2=symbolicmultiply(P_Num,C_Num);

if numel(kp1)>numel(kp2)
```

```
    kp=kp1 + [ zeros(1,numel(kp1)-numel(kp2)) kp2 ];
elseif numel(kp1)<numel(kp2)
    kp=kp2 + [ zeros(1,numel(kp2)-numel(kp1)) kp1 ];
else
    kp=kp1+kp2;                          % Characteristic Polynomial
end
de=poly(CL_Poles);
nkp=numel(kp);
nde=numel(de);
n=nkp-nde+1;
D=[];
P=[];
for i=1:n
D=[D [zeros(1,i-1) de zeros(1,nkp-nde-i+1)]'] ;   % D matrice
end
d=[];
e=[];
for i=1:numel(c)                         % Decompose symbolic and
real coefficients
 if findsym(c(i))
 d=[d i];
else
  e=[e i];
end
end
c=c(d);
c=c(1,1:numel(CL_Poles));
for i=c

   P=[ P  [(kp-subs(kp,i,0))/i].'];     % Get P matrice
end
A=kp.'-P*c.';                            % Get A matrice
sol_1=c.';
sol_2=inv([D -P])*A;                    % Solution
sol_3=sol_2(1:n,1);
sol_2=sol_2((n+1):numel(A),1);

konkat=sym2poly(sol_2);
u=1;
for i=d
    f(i)=konkat(u);                      % Place Controller
Coefficients
    u=u+1;
end
C_Num=double(f(1:numel(C_Num)));
C_Denum=double(f((numel(C_Num)+1):(numel(C_Num)+numel(C_Denum))));
Gc=tf(C_Num,C_Denum);                    % Transfer Function of the
Controller

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%
%   symbolicmultiply Function
%   Multiplies Symbolic Polynomials
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%

function y=symbolicmultiply(C_Num,C_Denum)
i=numel(C_Denum);
y=zeros(1,numel(C_Num)+numel(C_Denum)-1);
for n=1:i
    w = [1];
```

```
    for z=1:n
        if z~=1
        w=conv(w,[1 0]);
    end
    end

    y= y + [ zeros(1, numel(y)-numel(C_Denum(i-n+1)*conv(C_Num,w)))
C_Denum(i-n+1)*conv(C_Num,w) ];
end
```

APPENDIX C.3

dihsmf624.c

```
/* $Revision:  $ $Date:  $ */
/* dihsmf624.c - xPC Target, non-inlined S-function driver for
digital input section of Humusoft MF614 board  */
/* Copyright (c) 2003-2006 by Humusoft s.r.o. All Rights Reserved.
*/

#define     S_FUNCTION_LEVEL    2
#define     S_FUNCTION_NAME     dihsmf624

#include    <stddef.h>
#include    <stdlib.h>

#include    "simstruc.h"

#ifdef      MATLAB_MEX_FILE
#include    "mex.h"
#else
#include    <windows.h>
#include    "io_xpcimport.h"
#include    "pci_xpcimport.h"
#endif

/* Input Arguments */
#define NUM_PARAMS              (4)
#define SLOT_ARG                (ssGetSFcnParam(S,0))
#define DEV_ARG                 (ssGetSFcnParam(S,1))
#define CHANNEL_ARG             (ssGetSFcnParam(S,2))
#define SAMPLE_TIME_PARAM       (ssGetSFcnParam(S,3))

/* Convert S Function Parameters to Varibles */

#define SLOT                    ((uint_T) mxGetPr(SLOT_ARG)[0])
#define SAMPLE_TIME             ((real_T)
mxGetPr(SAMPLE_TIME_PARAM)[0])
#define SAMPLE_OFFSET           ((real_T)
mxGetPr(SAMPLE_TIME_PARAM)[1])
#define SAMP_TIME_IND           (0)

#define NO_I_WORKS              (2)
```

```
#define BASE_CHIPSET_I_IND     (0)
#define BASE_ADC_I_IND         (1)

#define NO_R_WORKS             (0)

static char_T msg[256];


/*====================*
 * S-function methods *
 *====================*/

static void mdlCheckParameters(SimStruct *S)
{
}

static void mdlInitializeSizes(SimStruct *S)
{
    uint_T i;

#ifndef MATLAB_MEX_FILE
#include "io_xpcimport.c"
#include "pci_xpcimport.c"
#endif


    ssSetNumSFcnParams(S, NUM_PARAMS);
    if (ssGetNumSFcnParams(S) == ssGetSFcnParamsCount(S)) {
        mdlCheckParameters(S);
        if (ssGetErrorStatus(S) != NULL) {
            return;
        }
    } else {
        return; /* Parameter mismatch will be reported by Simulink
*/
    }

    ssSetNumContStates(S, 0);
    ssSetNumDiscStates(S, 0);



    if (!ssSetNumInputPorts(S, 0)) return;

    if (!ssSetNumOutputPorts(S, mxGetNumberOfElements(CHANNEL_ARG)))
return;

    for (i=0;i<mxGetNumberOfElements(CHANNEL_ARG);i++) {
        ssSetOutputPortWidth(S, i, 1);
    }

    ssSetNumSampleTimes(S, 1);
    ssSetNumRWork(S, NO_R_WORKS);
    ssSetNumIWork(S, NO_I_WORKS);
    ssSetNumPWork(S, 0);

    ssSetNumModes(S, 0);
    ssSetNumNonsampledZCs(S, 0);
```

```
    ssSetSFcnParamNotTunable(S,0);
    ssSetSFcnParamNotTunable(S,1);
    ssSetSFcnParamNotTunable(S,2);
    ssSetSFcnParamNotTunable(S,3);

    ssSetOptions(S, SS_OPTION_EXCEPTION_FREE_CODE |
SS_OPTION_PLACE_ASAP);

}



static void mdlInitializeSampleTimes(SimStruct *S)
{
    ssSetSampleTime(S, 0, SAMPLE_TIME);
    ssSetOffsetTime(S, 0, SAMPLE_OFFSET);

}

#define MDL_START
static void mdlStart(SimStruct *S)
  {
#ifndef MATLAB_MEX_FILE
    PCIDeviceInfo pciinfo;
    void *Physical1, *Physical0;
    void *Virtual1, *Virtual0;
    volatile unsigned int *base_chipset;
    volatile unsigned short *base_adc;
    char devName[20];
    int  devId;

    switch ((int_T)mxGetPr(DEV_ARG)[0]) {
        case 1:
            strcpy(devName,"Humusoft MF624");
            devId=0x0624;
            break;
        case 2:
            strcpy(devName,"Humusoft AD622");
            devId=0x0622;
            break;
         }

    if ((int_T)mxGetPr(SLOT_ARG)[0]<0) {
        // look for the PCI-Device
        if (rl32eGetPCIInfo(0x186C,(unsigned short)devId,&pciinfo))
{
            sprintf(msg,"%s: board not present", devName);
            ssSetErrorStatus(S,msg);
            return;
        }
    } else {
        int_T bus, slot;
        if (mxGetN(SLOT_ARG) == 1) {
            bus  = 2; //0;
            slot = (int_T)mxGetPr(SLOT_ARG)[0];
        } else {
            bus = 2; //(int_T)mxGetPr(SLOT_ARG)[0];
            slot = (int_T)mxGetPr(SLOT_ARG)[1];
```

```
        }
        // look for the PCI-Device
        if (rl32eGetPCIInfoAtSlot(0x186C,(unsigned short)devId,(slot
& 0xff) | ((bus & 0xff)<< 8),&pciinfo)) {
            sprintf(msg,"%s (bus %d, slot %d): board not
present",devName, bus, slot );
            ssSetErrorStatus(S,msg);
            return;
        }
    }

    // show Device Information
    // rl32eShowPCIInfo(pciinfo);

    Physical0=(void *)pciinfo.BaseAddress[0];
    Virtual0 = rl32eGetDevicePtr(Physical0, 32,
RT_PG_USERREADWRITE);
    base_chipset=(void *)Physical0;
    Physical1=(void *)pciinfo.BaseAddress[2];
    Virtual1 = rl32eGetDevicePtr(Physical1, 128,
RT_PG_USERREADWRITE);
    base_adc=(void *)Physical1;

    ssSetIWorkValue(S, BASE_CHIPSET_I_IND, (int_T)base_chipset);
    ssSetIWorkValue(S, BASE_ADC_I_IND, (int_T)base_adc);

#endif /* MATLAB_MEX_FILE */
  }

static void mdlOutputs(SimStruct *S, int_T tid)
{
#ifndef MATLAB_MEX_FILE

    uint_T i;
    real_T *y;
    uint_T tempPortData;
    volatile unsigned short *base_adc;

    base_adc = (void *) ssGetIWorkValue(S, BASE_ADC_I_IND);

 /* Read in Digital Input from Hardware */

    tempPortData = base_adc[0x08];

    for (i=0;i<mxGetNumberOfElements(CHANNEL_ARG);i++) {
        y=ssGetOutputPortSignal(S,i);
        y[0] = (tempPortData >> (((short)mxGetPr(CHANNEL_ARG)[i])-
1))&0x01;
    }
#endif /* MATLAB_MEX_FILE */
}

static void mdlTerminate(SimStruct *S)
{
#ifndef MATLAB_MEX_FILE
#endif /* MATLAB_MEX_FILE */
}

/*============================*
```

```
 * Required S-function trailer *
 *=============================*/

#ifdef  MATLAB_MEX_FILE    /* Is this file being compiled as a MEX-
file? */
#include "simulink.c"      /* MEX-file interface mechanism */
#else
#include "cg_sfun.h"       /* Code generation registration function
*/
#endif
```

dohsmf624.c

```
/* $Revision:  $ $Date:  $ */
/* dohsmf624.c - xPC Target, non-inlined S-function driver for
digital output section of HUMUSOFT MF624 board */
/* Copyright (c) 2003-2006 by Humusoft s.r.o. All Rights Reserved.
*/

#define     S_FUNCTION_LEVEL    2
#define     S_FUNCTION_NAME     dohsmf624

#include    <stddef.h>
#include    <stdlib.h>

#include    "simstruc.h"

#ifdef      MATLAB_MEX_FILE
#include    "mex.h"
#else
#include    <windows.h>
#include    "io_xpcimport.h"
#include    "pci_xpcimport.h"
#endif


/* Input Arguments */
#define NUM_PARAMS              (4)
#define SLOT_ARG                (ssGetSFcnParam(S,0))
#define DEV_ARG                 (ssGetSFcnParam(S,1))
#define CHANNEL_ARG             (ssGetSFcnParam(S,2))
#define SAMPLE_TIME_PARAM       (ssGetSFcnParam(S,3))

/* Convert S Function Parameters to Varibles */

#define SLOT                    ((uint_T) mxGetPr(SLOT_ARG)[0])
#define SAMPLE_TIME             ((real_T)
mxGetPr(SAMPLE_TIME_PARAM)[0])
#define SAMPLE_OFFSET           ((real_T)
mxGetPr(SAMPLE_TIME_PARAM)[1])
#define SAMP_TIME_IND           (0)

#define NO_I_WORKS              (2)
#define BASE_CHIPSET_I_IND      (0)
#define BASE_ADC_I_IND          (1)

#define NO_R_WORKS              (0)

static char_T msg[256];
```

```
/*====================*
 * S-function methods *
 *====================*/

static void mdlCheckParameters(SimStruct *S)
{
}

static void mdlInitializeSizes(SimStruct *S)
{
    uint_T i;

#ifndef MATLAB_MEX_FILE
#include "io_xpcimport.c"
#include "pci_xpcimport.c"
#endif

    ssSetNumSFcnParams(S, NUM_PARAMS);
    if (ssGetNumSFcnParams(S) == ssGetSFcnParamsCount(S)) {
        mdlCheckParameters(S);
        if (ssGetErrorStatus(S) != NULL) {
            return;
        }
    } else {
        return; /* Parameter mismatch will be reported by Simulink
*/
    }

    ssSetNumContStates(S, 0);
    ssSetNumDiscStates(S, 0);

    if (!ssSetNumInputPorts(S, mxGetNumberOfElements(CHANNEL_ARG)))
return;

    for (i=0;i<mxGetNumberOfElements(CHANNEL_ARG);i++) {
        ssSetInputPortWidth(S, i, 1);
    }

    if (!ssSetNumOutputPorts(S, 0)) return;

    ssSetNumSampleTimes(S, 1);
    ssSetNumRWork(S, NO_R_WORKS);
    ssSetNumIWork(S, NO_I_WORKS);
    ssSetNumPWork(S, 0);

    ssSetNumModes(S, 0);
    ssSetNumNonsampledZCs(S, 0);

    ssSetSFcnParamNotTunable(S,0);
    ssSetSFcnParamNotTunable(S,1);
    ssSetSFcnParamNotTunable(S,2);
    ssSetSFcnParamNotTunable(S,3);

    ssSetOptions(S, SS_OPTION_EXCEPTION_FREE_CODE |
SS_OPTION_PLACE_ASAP);
```

```
}

static void mdlInitializeSampleTimes(SimStruct *S)
{
    ssSetSampleTime(S, 0, SAMPLE_TIME);
    ssSetOffsetTime(S, 0, SAMPLE_OFFSET);

}


#define MDL_START
static void mdlStart(SimStruct *S)
{

#ifndef MATLAB_MEX_FILE
    PCIDeviceInfo pciinfo;
    void *Physical1, *Physical0;
    void *Virtual1, *Virtual0;
    volatile unsigned int *base_chipset;
    volatile unsigned short *base_adc;
    char devName[20];
    int  devId;

    switch ((int_T)mxGetPr(DEV_ARG)[0]) {
        case 1:
            strcpy(devName,"Humusoft MF624");
            devId=0x0624;
            break;
        case 2:
            strcpy(devName,"Humusoft AD622");
            devId=0x0622;
            break;
         }

    if ((int_T)mxGetPr(SLOT_ARG)[0]<0) {
        // look for the PCI-Device
        if (rl32eGetPCIInfo(0x186C,(unsigned short)devId,&pciinfo))
{
            sprintf(msg,"%s: board not present", devName);
            ssSetErrorStatus(S,msg);
            return;
        }
    } else {
        int_T bus, slot;
        if (mxGetN(SLOT_ARG) == 1) {
            bus  = 2; //0;
            slot = (int_T)mxGetPr(SLOT_ARG)[0];
        } else {
            bus = 2; //(int_T)mxGetPr(SLOT_ARG)[0];
            slot = (int_T)mxGetPr(SLOT_ARG)[1];
        }
        // look for the PCI-Device
        if (rl32eGetPCIInfoAtSlot(0x186C,(unsigned short)devId,(slot
& 0xff) | ((bus & 0xff)<< 8),&pciinfo)) {
            sprintf(msg,"%s (bus %d, slot %d): board not
present",devName, bus, slot );
            ssSetErrorStatus(S,msg);
            return;
        }
    }
```

```
    // show Device Information
    // rl32eShowPCIInfo(pciinfo);

    Physical0=(void *)pciinfo.BaseAddress[0];
    Virtual0 = rl32eGetDevicePtr(Physical0, 32,
RT_PG_USERREADWRITE);
    base_chipset=(void *)Physical0;
    Physical1=(void *)pciinfo.BaseAddress[2];
    Virtual1 = rl32eGetDevicePtr(Physical1, 128,
RT_PG_USERREADWRITE);
    base_adc=(void *)Physical1;

    ssSetIWorkValue(S, BASE_CHIPSET_I_IND, (int_T)base_chipset);
    ssSetIWorkValue(S, BASE_ADC_I_IND, (int_T)base_adc);

    /* Set Digital Output to Initial State */

    base_adc[0x08] = 0;

#endif /* MATLAB_MEX_FILE */

}

static void mdlOutputs(SimStruct *S, int_T tid)
{

#ifndef MATLAB_MEX_FILE

    uint_T i;
    InputRealPtrsType uPtrs;
    uint_T mf624DOPortData = 0;
    volatile unsigned short *base_adc;

    base_adc = (void *) ssGetIWorkValue(S, BASE_ADC_I_IND);

    /* Send Input to Digital Output */
    mf624DOPortData = 0;

    for (i=0;i < mxGetNumberOfElements(CHANNEL_ARG);i++) {
        uPtrs = ssGetInputPortRealSignalPtrs(S,i);
        mf624DOPortData |= ((short)*uPtrs[0] <<
(((short)mxGetPr(CHANNEL_ARG)[i])-1));
    }

    base_adc[0x08] = mf624DOPortData;

#endif /* METLAB_MEX_FILE */
}

static void mdlTerminate(SimStruct *S)
{
#ifndef MATLAB_MEX_FILE

    volatile unsigned short *base_adc;

    base_adc = (void *) ssGetIWorkValue(S, BASE_ADC_I_IND);
```

```
    /* Set Final State of digital output */

    base_adc[0x08] = 0;

    ssSetUserData(S,NULL);

#endif /* MATLAB_MEX_FILE */
}

/*=============================*
 * Required S-function trailer *
 *=============================*/

#ifdef  MATLAB_MEX_FILE    /* Is this file being compiled as a MEX-
file? */
#include "simulink.c"      /* MEX-file interface mechanism */
#else
#include "cg_sfun.h"       /* Code generation registration function
*/
#endif
```

enchsmf624.c

```
/* $Revision:  $ $Date:  $ */
/* enchsmf624.c - xPC Target, non-inlined S-function driver for
encoder section of Humusoft MF624 board  */
/* Copyright (c) 2003-2006 by Humusoft s.r.o. All Rights Reserved.
*/

#define     S_FUNCTION_LEVEL    2
#define     S_FUNCTION_NAME     enchsmf624

#include    <stddef.h>
#include    <stdlib.h>

#include    "simstruc.h"

#ifdef      MATLAB_MEX_FILE
#include    "mex.h"
#else
#include    <windows.h>
#include    "io_xpcimport.h"
#include    "pci_xpcimport.h"
#endif


/* Input Arguments */
#define NUM_PARAMS              (6)
#define SLOT_ARG                (ssGetSFcnParam(S,0))
#define DEV_ARG                 (ssGetSFcnParam(S,1))
#define CHANNEL_ARG             (ssGetSFcnParam(S,2))
#define FILTER_ARG              (ssGetSFcnParam(S,3))
#define INDEX_ARG               (ssGetSFcnParam(S,4))
#define SAMPLE_TIME_PARAM       (ssGetSFcnParam(S,5))

/* Convert S Function Parameters to Varibles */

#define SLOT                    ((uint_T) mxGetPr(SLOT_ARG)[0])
```

```
#define SAMPLE_TIME             ((real_T)
mxGetPr(SAMPLE_TIME_PARAM)[0])
#define SAMPLE_OFFSET           ((real_T)
mxGetPr(SAMPLE_TIME_PARAM)[1])
#define SAMP_TIME_IND           (0)

#define NO_I_WORKS              (3)
#define BASE_CHIPSET_I_IND      (0)
#define BASE_ADC_I_IND          (1)
#define BASE_ALTERA_I_IND       (2)

#define NO_R_WORKS              (0)

#define MAXIRCFILT 2500000

static char_T msg[256];


/*====================*
 * S-function methods *
 *====================*/

static void mdlCheckParameters(SimStruct *S)
{
}

static void mdlInitializeSizes(SimStruct *S)
{
    uint_T i;

#ifndef MATLAB_MEX_FILE
#include "io_xpcimport.c"
#include "pci_xpcimport.c"
#endif


    ssSetNumSFcnParams(S, NUM_PARAMS);
    if (ssGetNumSFcnParams(S) == ssGetSFcnParamsCount(S)) {
        mdlCheckParameters(S);
        if (ssGetErrorStatus(S) != NULL) {
            return;
        }
    } else {
        return; /* Parameter mismatch will be reported by Simulink
*/
    }

    ssSetNumContStates(S, 0);
    ssSetNumDiscStates(S, 0);



    if (!ssSetNumInputPorts(S, 0)) return;

    if (!ssSetNumOutputPorts(S, mxGetNumberOfElements(CHANNEL_ARG)))
return;

    for (i=0;i<mxGetNumberOfElements(CHANNEL_ARG);i++) {
```

```
        ssSetOutputPortWidth(S, i, 1);
    }

    ssSetNumSampleTimes(S, 1);
    ssSetNumRWork(S, NO_R_WORKS);
    ssSetNumIWork(S, NO_I_WORKS);
    ssSetNumPWork(S, 0);

    ssSetNumModes(S, 0);
    ssSetNumNonsampledZCs(S, 0);

    ssSetSFcnParamNotTunable(S,0);
    ssSetSFcnParamNotTunable(S,1);
    ssSetSFcnParamNotTunable(S,2);
    ssSetSFcnParamNotTunable(S,3);
    ssSetSFcnParamNotTunable(S,4);
    ssSetSFcnParamNotTunable(S,5);

    ssSetOptions(S, SS_OPTION_EXCEPTION_FREE_CODE |
SS_OPTION_PLACE_ASAP);

}



static void mdlInitializeSampleTimes(SimStruct *S)
{
    ssSetSampleTime(S, 0, SAMPLE_TIME);
    ssSetOffsetTime(S, 0, SAMPLE_OFFSET);
}



#define MDL_START
static void mdlStart(SimStruct *S)
{

#ifndef MATLAB_MEX_FILE

    uint_T i, channel;
    PCIDeviceInfo pciinfo;
    void *Physical2, *Physical1, *Physical0;
    void *Virtual2, *Virtual1, *Virtual0;
    volatile unsigned int *base_chipset;
    volatile unsigned short *base_adc;
    volatile unsigned int *base_altera;
    char devName[20];
    int  devId;
    unsigned mode;
    unsigned irc_shadow;

    switch ((int_T)mxGetPr(DEV_ARG)[0]) {
        case 1:
            strcpy(devName,"Humusoft MF624");
            devId=0x0624;
            break;
         }
```

```
    if ((int_T)mxGetPr(SLOT_ARG)[0]<0) {
        // look for the PCI-Device
        if (rl32eGetPCIInfo(0x186C,(unsigned short)devId,&pciinfo))
{
            sprintf(msg,"%s: board not present", devName);
            ssSetErrorStatus(S,msg);
            return;
        }
    } else {
        int_T bus, slot;
        if (mxGetN(SLOT_ARG) == 1) {
            bus  = 2; //0;
            slot = (int_T)mxGetPr(SLOT_ARG)[0];
        } else {
            bus = 2; //(int_T)mxGetPr(SLOT_ARG)[0];
            slot = (int_T)mxGetPr(SLOT_ARG)[1];
        }
        // look for the PCI-Device
        if (rl32eGetPCIInfoAtSlot(0x186C,(unsigned short)devId,(slot
& 0xff) | ((bus & 0xff)<< 8),&pciinfo)) {
            sprintf(msg,"%s (bus %d, slot %d): board not
present",devName, bus, slot );
            ssSetErrorStatus(S,msg);
            return;
        }
    }

    // show Device Information
    // rl32eShowPCIInfo(pciinfo);

    Physical0=(void *)pciinfo.BaseAddress[0];
    Virtual0 = rl32eGetDevicePtr(Physical0, 32,
RT_PG_USERREADWRITE);
    base_chipset=(void *)Physical0;
    Physical1=(void *)pciinfo.BaseAddress[2];
    Virtual1 = rl32eGetDevicePtr(Physical1, 128,
RT_PG_USERREADWRITE);
    base_adc=(void *)Physical1;
    Physical2=(void *)pciinfo.BaseAddress[4];
    Virtual2 = rl32eGetDevicePtr(Physical2, 128,
RT_PG_USERREADWRITE);
    base_altera=(void *)Physical2;

    ssSetIWorkValue(S, BASE_CHIPSET_I_IND, (int_T)base_chipset);
    ssSetIWorkValue(S, BASE_ADC_I_IND, (int_T)base_adc);
    ssSetIWorkValue(S, BASE_ALTERA_I_IND, (int_T)base_altera);

    base_altera[0x1B] = 0x10101010;
// set x4 mode, reset counters
    base_altera[0x1B] = 0x0;
// set x4 mode, enable counters
    irc_shadow = 0;

//    for (i=0;i<mxGetNumberOfElements(CHANNEL_ARG);i++) {
//        channel=(uint_T)mxGetPr(CHANNEL_ARG)[i]-1;
//
//        mode = 0;
//        irc_shadow |= (mxGetPr(FILTER_ARG)[i] && 0x01) << 8 *
channel;
//        switch ((uint_T)mxGetPr(INDEX_ARG)[i]) {
```

```
//          case 0:    mode = 0x00; break;
//          case 1:    mode = 0x20; break;
//          case 2:    mode = 0x30; break;
//          case 3:    mode = 0x40; break;
//          case 4:    mode = 0x50; break;
//          case 5:    mode = 0x60; break;
//          case 6:    mode = 0x0C; break;
//          case 7:    mode = 0x08; break;
//      }
//        irc_shadow |= mode << 8 * channel;
//     }
//    base_altera[0x1B] = irc_shadow;

#endif

}

static void mdlOutputs(SimStruct *S, int_T tid)
{

#ifndef MATLAB_MEX_FILE

    int_T i, n, channel, irc;
    volatile int *base_altera;
    real_T *y;

    base_altera = (void *) ssGetIWorkValue(S, BASE_ALTERA_I_IND);

    for (i=0;i<mxGetNumberOfElements(CHANNEL_ARG);i++) {
        channel=(uint_T)mxGetPr(CHANNEL_ARG)[i]-1;
        y=ssGetOutputPortSignal(S,i);

        y[0]=(real_T) base_altera[0x1C+channel];
    }

#endif

}

static void mdlTerminate(SimStruct *S)
{
}


#ifdef MATLAB_MEX_FILE  /* Is this file being compiled as a MEX-
file? */
#include "simulink.c"   /* Mex glue */
#else
#include "cg_sfun.h"    /* Code generation glue */
#endif
```

dahsmf624.c

```
/* $Revision:  $ $Date:  $ */
/* dahsmf624.c - xPC Target, non-inlined S-function driver for
Humusoft MF624 D/A section */
/* Copyright (c) 2003-2006 by Humusoft s.r.o. All Rights Reserved.
*/
```

```
#define S_FUNCTION_LEVEL 2
#define S_FUNCTION_NAME  dahsmf624

#include    <stddef.h>
#include    <stdlib.h>
#include    <math.h>

#include    "simstruc.h"

#ifdef      MATLAB_MEX_FILE
#include    "mex.h"
#else
#include    <windows.h>
#include    "io_xpcimport.h"
#include    "pci_xpcimport.h"
#endif

/* Input Arguments */
#define NUM_PARAMS              (5)
#define SLOT_ARG                (ssGetSFcnParam(S,0))
#define DEV_ARG                 (ssGetSFcnParam(S,1))
#define CHANNEL_ARG             (ssGetSFcnParam(S,2))
#define RANGE_ARG               (ssGetSFcnParam(S,3))
#define SAMPLE_TIME_PARAM       (ssGetSFcnParam(S,4))

/* Convert S Function Parameters to Varibles */

#define SLOT                    ((uint_T) mxGetPr(SLOT_ARG)[0])
#define DEVICE                  ((uint_T) mxGetPr(DEV_ARG)[0])
#define SAMPLE_TIME             ((real_T)
mxGetPr(SAMPLE_TIME_PARAM)[0])
#define SAMPLE_OFFSET           ((real_T)
mxGetPr(SAMPLE_TIME_PARAM)[1])
#define SAMP_TIME_IND           (0)

#define NO_I_WORKS              (3)
#define BASE_CHIPSET_I_IND      (0)
#define BASE_ADC_I_IND          (1)
#define BASE_ALTERA_I_IND       (2)

#define NO_R_WORKS              (0)

static char_T msg[256];


#ifndef MATLAB_MEX_FILE
static void DA_Output(SimStruct *S, uint_T device, uint_T channel,
real_T value)
{
    real_T out;
    volatile unsigned int *base_chipset;
    volatile unsigned short *base_adc;

    base_chipset  = (void *) ssGetIWorkValue(S, BASE_CHIPSET_I_IND);
    base_adc = (void *) ssGetIWorkValue(S, BASE_ADC_I_IND);

    out=8192*(1-value/-10);
```

```
    out=max(out,0);
    out=min(out,16383);
    out=floor(out+0.5);

    base_adc[0x10+channel] =(uint_T)out;
}
#endif

/*====================*
 * S-function methods *
 *====================*/

static void mdlCheckParameters(SimStruct *S)
{
}

static void mdlInitializeSizes(SimStruct *S)
{
    uint_T i;

#ifndef MATLAB_MEX_FILE
#include "io_xpcimport.c"
#include "pci_xpcimport.c"
#endif

    ssSetNumSFcnParams(S, NUM_PARAMS);
    if (ssGetNumSFcnParams(S) == ssGetSFcnParamsCount(S)) {
        mdlCheckParameters(S);
        if (ssGetErrorStatus(S) != NULL) {
            return;
        }
    } else {
        return; /* Parameter mismatch will be reported by Simulink
*/
    }

    ssSetNumContStates(S, 0);
    ssSetNumDiscStates(S, 0);

    if (!ssSetNumInputPorts(S, mxGetNumberOfElements(CHANNEL_ARG)))
return;

    for (i=0;i<mxGetNumberOfElements(CHANNEL_ARG);i++) {
        ssSetInputPortWidth(S, i, 1);
    }

    if (!ssSetNumOutputPorts(S, 0)) return;

    ssSetNumSampleTimes(S, 1);
    ssSetNumRWork(S, NO_R_WORKS);
    ssSetNumIWork(S, NO_I_WORKS);
    ssSetNumPWork(S, 0);

    ssSetNumModes(S, 0);
    ssSetNumNonsampledZCs(S, 0);

    ssSetSFcnParamNotTunable(S,0);
    ssSetSFcnParamNotTunable(S,1);
```

```
    ssSetSFcnParamNotTunable(S,2);
    ssSetSFcnParamNotTunable(S,3);
    ssSetSFcnParamNotTunable(S,4);

    ssSetOptions(S, SS_OPTION_EXCEPTION_FREE_CODE |
SS_OPTION_PLACE_ASAP);
}

static void mdlInitializeSampleTimes(SimStruct *S)
{
    ssSetSampleTime(S, 0, SAMPLE_TIME);
    ssSetOffsetTime(S, 0, SAMPLE_OFFSET);
}


#define MDL_START
static void mdlStart(SimStruct *S)
{

#ifndef MATLAB_MEX_FILE
    uint_T i;
    PCIDeviceInfo pciinfo;
    void *Physical1, *Physical0;
    void *Virtual1, *Virtual0;
    volatile unsigned int *base_chipset;
    volatile unsigned short *base_adc;
    char devName[20];
    int  devId;

    switch ((int_T)mxGetPr(DEV_ARG)[0]) {
        case 1:
            strcpy(devName,"Humusoft MF624");
            devId=0x0624;
            break;
        case 2:
            strcpy(devName,"Humusoft AD622");
            devId=0x0622;
            break;
         }

    if ((int_T)mxGetPr(SLOT_ARG)[0]<0) {
        // look for the PCI-Device
        if (rl32eGetPCIInfo(0x186C,(unsigned short)devId,&pciinfo))
{
            sprintf(msg,"%s: board not present", devName);
            ssSetErrorStatus(S,msg);
            return;
        }
    } else {
        int_T bus, slot;
        if (mxGetN(SLOT_ARG) == 1) {
            bus  = 2; //0;
            slot = (int_T)mxGetPr(SLOT_ARG)[0];
        } else {
            bus = 2; //(int_T)mxGetPr(SLOT_ARG)[0];
            slot = (int_T)mxGetPr(SLOT_ARG)[1];
        }
        // look for the PCI-Device
        if (rl32eGetPCIInfoAtSlot(0x186C,(unsigned short)devId,(slot
& 0xff) | ((bus & 0xff)<< 8),&pciinfo)) {
```

```
            sprintf(msg,"%s (bus %d, slot %d): board not
present",devName, bus, slot );
            ssSetErrorStatus(S,msg);
            return;
        }
    }

    // show Device Information
    // rl32eShowPCIInfo(pciinfo);

    Physical0=(void *)pciinfo.BaseAddress[0];
    Virtual0 = rl32eGetDevicePtr(Physical0, 32,
RT_PG_USERREADWRITE);
    base_chipset=(void *)Physical0;
    Physical1=(void *)pciinfo.BaseAddress[2];
    Virtual1 = rl32eGetDevicePtr(Physical1, 128,
RT_PG_USERREADWRITE);
    base_adc=(void *)Physical1;

    ssSetIWorkValue(S, BASE_CHIPSET_I_IND, (int_T)base_chipset);
    ssSetIWorkValue(S, BASE_ADC_I_IND, (int_T)base_adc);

/* Set initial outputs to zero */
    if (!(base_chipset[0x15] & 0x04000000));  //Nobody enabled DACEN
yet, set 0V. DAC will be enabled after 1st conversion.
        for (i=0;i<8;i++) {
            DA_Output(S, DEVICE, i, 0);
        }

#endif /* MATLAB_MEX_FILE */

}

static void mdlOutputs(SimStruct *S, int_T tid)
{
#ifndef MATLAB_MEX_FILE

    InputRealPtrsType uPtrs;
    uint_T i;
    volatile unsigned int *base_chipset;

    base_chipset  = (void *) ssGetIWorkValue(S, BASE_CHIPSET_I_IND);
    base_chipset[0x15] = 0x06C006C0;        // disable update

    for (i=0;i<mxGetNumberOfElements(CHANNEL_ARG);i++) {
        uPtrs = ssGetInputPortRealSignalPtrs(S,i);
        DA_Output(S, DEVICE, (uint_T)mxGetPr(CHANNEL_ARG)[i]-1,
*uPtrs[0]);
    }

    base_chipset[0x15] = 0x064006C0;       // simultaneous update

#endif /* MATLAB_MEX_FILE */

}


static void mdlTerminate(SimStruct *S)
```

```
{
#ifndef MATLAB_MEX_FILE
    uint_T i;
    volatile unsigned int *base_chipset;

    base_chipset  = (void *) ssGetIWorkValue(S, BASE_CHIPSET_I_IND);
    base_chipset[0x15] = 0x068006C0;        // disable update

/* Set final outputs to zero */
    for (i=0;i < mxGetNumberOfElements(CHANNEL_ARG);i++) {
        DA_Output(S, DEVICE, (uint_T)mxGetPr(CHANNEL_ARG)[i]-1, 0);
    }

    base_chipset[0x15] = 0x064006C0;       // simultaneous update

#endif /* MATLAB_MEX_FILE */
}

#ifdef  MATLAB_MEX_FILE    /* Is this file being compiled as a MEX-
file? */
#include "simulink.c"      /* MEX-file interface mechanism */
#else
#include "cg_sfun.h"       /* Code generation registration function
*/
#endif
```

CURRICULUM VITA

Candidate's full name: Cüneyt KARACA

Place and date of birth: KARABÜK, 1983

Permanent Address: Tepebaşı Mah. Diyar Sok. 15/1 KEÇİÖREN/ANKARA

Universities and

Colleges attended : İstanbul Technical University, Control Engineering