MATLAB based real time control implementation of DC servo motor using PCI card

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF

THE REQUIREMENTS FOR THE DEGREE OF

Bachelor of Technology

in

Electrical Engineering

By

Ananya Roy (107EE025)
Aditya Gazta (107EE029)
Suneet Sahadevan (107EE013)

Department of Electrical Engineering

National Institute of Technology

Rourkela

2010-2011

CERTIFICATE

This is to certify That this thesis entitled "**MATLAB based real time control implementation of DC servo motor using PCI card**" submitted by Ananya Roy, Aditya Gazta and Suneet Sahadevan in partial fulfillment of the requirements for the award of Bachelor of Technology degree in Electrical Engineering at National Institute of Technology Rourkela (Deemed University), is an authentic work carried out by him under my supervision and guidance.

To the best of my knowledge the matter embodied in the thesis has not been submitted to any other University/Institute for the award of any degree or diploma.

Date:

Place: Rourkela

Prof. Sandip Ghosh

Department of Electrical Engineering

National Institute of Technology Rourkela

ABSTRACT

The DC servo motor has applications in automotive market in applications ranging from heating and ventilation to power mirror positioning. It is also in use in Industrial and Consumer Markets for entertainment equipment, HVAC ventilation control and myriad number of other applications.

In all these applications, either speed or position control of the DC servo motor is used.

In this project, we have achieved MATLAB based real-time speed control implementation of DC servo motor using PCI-1716.

For a more efficient speed control, closed loop control system of the servo motor is realized with the help of a tuned PID controller.

Further, progress has also been made in remote control of DC servo motor. For this, a communication has been established between the local and the remote PC; and PID controlled closed loop system, with the controller at the remote PC was successfully achieved and checked for optimal performance.

However, due to delays introduced in the communication system, the remote control using PID was not found to be as efficient as the local control and the speed output was found to lag behind the reference signal. To compensate for this delay, a Smith Predictor incorporated model has been made.

ACKNOWLEDGEMENT

We would like to take this opportunity to express my gratitude and sincere thanks to our respected supervisor Prof. Sandip Gosh for his guidance, insight, and support he has provided throughout the course of this work.

We would like to thank all faculty members and staff of the Department of Electrical Engineering, N.I.T. Rourkela for their help throughout the course.

We would specially like to thank Mr. Srinibas Bhuiyan (M.Tech research student) for his time and help with our project.

Ananya Roy (107EE025)
Aditya Gazta (107EE029)
Suneet Sahadevan (107EE013)

Contents

List of Figures

List of Tables

Chapter 1. Introduction

1.1 Motivation

The various steps which have been undertaken in this project are:

1. Interfacing between PCI card and computer through which we will control the dc servo motor
2. Interfacing between two computers using UDP protocol
3. Modeling the transfer function of the servo motor using speed control
4. Designing of PID controller for the transfer function
5. Tuning of PID controller using pole-placement method
6. Motor control through PID controller.
7. Remote control using UDP protocol

1.2 The Servo System

A closed-loop control system is another name for a servo system. To be classified as a servo, a control system must be capable of the following:

1. Accepting an order that defines the desired result.
2. Determining the present conditions by some method of feedback.
3. Comparing the desired result with the present conditions and obtaining a difference or an error signal.
4. Issuing a correcting order (the error signal) that will properly change the existing conditions to the desired result.
5. Obeying the correcting order.

- (Servo)

1.3 Servo Motor

A servo motor is an electric motor with a built in rotation sensor, they are needed for robotics. Say a robot moves its arm by turning a servo motor, the motor would send information concerning the degree of rotation on its axis back to the robot so the robot can keep tabs on the position of its arm, so if something bumps its arm it will know it and so-on.

Electric motors are the commonly used actuator in electromagnetic systems of all types. They are made in a variety of configurations and sizes for applications ranging from activating precision movements to powering diesel-electric locomotives. The laboratory motors are small servomotors, which might be used for positioning and speed control applications in a variety of automated machines. They are DC (direct current) motors. The armature is driven by an external DC voltage that produces the motor torque and results in the motor speed. The armature current produced by the applied voltage interacts with the permanent magnet field to produce current and motion. - (Modeling of DC Servo Motor)

The servo DC motor is basically a transducer that converts electric energy into mechanical energy. The torque developed on the motor shaft is directly proportional to the field flux and the armature current. The dc servo motor servo motors are very expensive in comparison to ac servo motors because of brushes and commutators .These motors have relatively low torque to volume and torque to inertia ratio, however the characteristics of dc motors are quite linear and are easy to control.

With the development of rare earth magnet dc motors which have high torque to volume ratio at reasonable cost. Advancement in technology has also lead to brushless dc motors

quite popular in high performance control system the sketch of the basic components of dc motor is given in fig below. The non-turning part called stator and has magnet which establishes a field across the rotor the turning part. The magnet may be electromagnet or permanent one. In electromagnet motor the stator has wire wound across and current through winding called field winding. For a constant field current $i_{f,}$ the magnetic flux is constant; the flux can be varied by varying the field current.

1.4 Control Aspects Of Servo Motor

An open-loop system has no feedback from the system being controlled. This is sort of shoot and forget approach to control. An input signal or value is provided and the controller commands the system to go to a definite speed, whatever, and hopes that the system responds accordingly. There is no information from the system under control to show that it even got the command or acted upon it.

In a closed-loop control system there is feedback. If speed is being controlled, a small amount of the current speed is provided back to the controller, allowing it to adjust its commands as the system responds to the commands. Likewise true with position.

The performance of a closed-loop system is partially a function of the speed at which the feedback is returned to the controller. This closing of the loop will always take a set amount of time, and the longer that time is the less responsive the controller will be to fast changing conditions.

Depending on the performance requirements of the application, either a closed-loop or open-loop control system can be used to control motor position, speed, or other similar application.- (Servo Motor Control)

1.5 Modeling Of The DC Servo Motor

The servo DC motor is used extensively in control systems, for analytical purpose, it is necessary to establish mathematical models for dc motors for applications. We use the equivalent circuit diagram in fig. to represent a dc motor. The armature is modeled as a circuit with resistance R_a Connected in series with an inductance L_a and a voltage source e_b representing the back elf in the armature when the rotor rotates. The motors variables and parameters are defined as follows:

With reference to the circuit diagram of the above figure, the control of the DC motor is applied at the armature terminals in the form of applied voltage $e_a(t)$. For linear analysis, we assume that the torque developed by the motor is proportional to the air-gap flux and the armature current. Thus,

$$T_m(t) = K_m(t)\phi i_e(t) \quad \text{.....(1)}$$

Since ϕ is constant, equation (1) is written

$$T_m(t) = K_i i_e(t)$$

Where K_i is the torque constant in N-m/A.

Starting with the control input voltage, $e_a(t)$ the cause-and-effect equations for the motor circuit in fig are:

$i_a(t) = armature\ current$ $R_a = armature\ resistance$

$e_b(t) = back\ emf$ $T_l(t) = load\ torque$

$T_m(t) = motor\ torque$ $\theta_m(t)$ = rotor displacement

$K_i = torque\ constant$ $L_a = armature\ inductance$

$e_a = applied\ voltage$ $K_b = back\ emf\ constant$

$\phi = magnetic\ flux\ in\ the\ air\ gap$ $\omega_m(t) = rotor\ angular\ velocity$

$J_m = rotor\ inertia$ $B_m = viscous - friction\ coefficient$

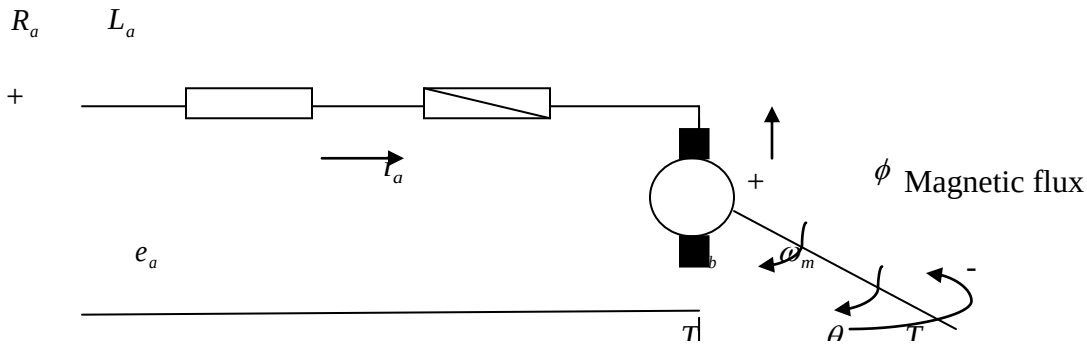

Fig 1.1: DC servo motor circuit diagram

The KVL equation for the above circuit:

$$\frac{di_a(t)}{dt} = \frac{1}{L_a}e_a(t) - \frac{R_a}{L_a}i_a(t) - \frac{1}{L_a}e_b(t)$$

The back-emf is given by:

$$e_b(t) = K_b \frac{d\theta_m(t)}{dt} = K_b \omega_m(t)$$

Combining these two equations and taking the Laplace transform,

$$I_a(s) = \frac{(E_a - K_b \omega_m)}{(R_a + sL_a)}$$

The torque equation:

$$T_m(t) = K_i i_a(t)$$

$$T_m(s) = K_i \times \frac{(E_a - K_b \omega_m)}{(R_a + sL_a)}$$

The mechanical equation:

$$\frac{d^2\theta_m(t)}{dt^2} = \frac{1}{J_m} T_m(t) - \frac{1}{J_m} T_L(t) - \frac{B_m}{J_m} \frac{d\theta_m(t)}{dt}$$

(Where $T_L(t)$ represents a load frictional torque such as coulomb friction)

$$s\omega_m = \left(\frac{1}{J_m}\right)\left[K_i\left(\frac{E_a - K_b \omega_m}{R_a + sL_a}\right)\right] - \frac{T_L}{J_m} - \frac{B_m \omega_m}{J_m}$$

$$\frac{\omega_m(s)}{e_a(s)} = \frac{K_i}{(J_m L_a s^2 + (J_m R_a + B_m R_a + B_m L_a)s + K_i K_b)}$$

This is the transfer function for the servo motor using the speed control method, where speed is the output and voltage as the input.

1.6 Applications

In Automotive Market:

1. Power mirror positioning.
2. Power seats positioning motors.
3. Power door and trunk lock mechanisms.
4. Windshield wiper motors.
5. Heating, Ventilation, and Air Conditioning (HVAC) vent controls.
6. Power sliding door, sunroof, and convertible top actuators.
7. Headlight positioning and leveling actuators.

In Industrial and Consumer Markets:

1. Proportioning valves for gasses and liquids.
2. Paper and materials handling equipment.
3. HVAC ventilation control.
4. Entertainment equipment (powered, remotely controlled volume controls for audio receivers and mixers)

Chapter 2. Data Acquisition Through The PCI Card (PCI 1716)

2.1 Introduction

PCI-1716 and PCI-1716L are powerful high-resolution multifunction cards for the PCI bus. They feature a 250 kS/s 16-bit A/D converter, and an onboard 1,024-sample FIFO buffer for A/D. The cards have up to 16 single-ended or 8 differential A/D input channels or a combination of these; two 16-bit D/A output channels, 16 digital input/output channels, and one 10 MHz 16-bit counter channel. PCI-1716 and PCI-1716L provide specific functions for different user requirements.

2.2 Specifications Of PCI 1716

Analog Input

Channels: 16 single-ended/ 8 differential (software programmable)

Resolution: 16 bits

Max. Sampling Rate: 250 kS/s

FIFO Size: 1,024 samples

Overvoltage Protection: 30 Vp-p

Input Impedance: 100 MΩ/10 pF (off), 100 MΩ/100 pF (on)

Sampling Modes: Software, onboard programmable pacer and external

Input Range (V, software programmable)

Table 2.1; Input specification of PCI card 1716

Unipolar	N/A	0-10	0-5	0-2.5	0-1.25
Bipolar	+10	+_5	2.5	1.25	0.625
Accuracy	0.05	0.03	0.03	0.05	0.1

Analog Output (PCI-1716 only)

Channels: 2

Resolution: 16 bits

Output Rate: Static update

Output Range (V, software programmable)

Slew Rate: 20 V/μs

Driving Capability: 20 mA

Output Impedance: 0.1 W max.

Operation Mode: Software polling

Accuracy INLE: ±1 LSB

Digital Input

Channels: 16

Compatibility: 5 V/TTL

Input Voltage: Logic 0: 0.8 V max. Logic 1: 2.0 V min.

Digital Output

Channels: 16

Compatibility: 5 V/TTL

Output Voltage: Logic 0: 0.4 V max. Logic 1: 2.4 V min.

Output Capability Sink: 0.8 mA @ 0.8 V Source: -2.4 mA @ 2.0 V

Pacer/Counter

Channels: 1

Resolution: 16 bits

Compatibility: 5 V/TTL

Max. Input Frequency: 1 MHz

Reference Clock Internal: 10 MHz External Clock Frequency: 10 MHz max.

General

Bus Type: PCI V2.2

I/O Connector: 1 x 68-pin SCSI female connector

Dimensions: (L x H) 175 x 100 mm (6.9" x 3.9")

Power Consumption: Typical: 5 V @ 850 mA, 12 V @ 600 mA Max.: 5 V @ 1 A, 12 V @ 700 mA

Operating Temperature: 0 ~ 70° C (32 ~ 158° F) (refer to IEC 68-2-1, 2)

Storage Temperature: -20 ~ 85° C (-4 ~ 185° F)

Operating Humidity: 5 ~ 85% RH non-condensing(refer to IEC 68-1, -2, -3)

Storage Humidity: 5 ~ 95% RH non-condensing (refer to IEC 68-1, -2, -3)

2.3 Pin Configuration

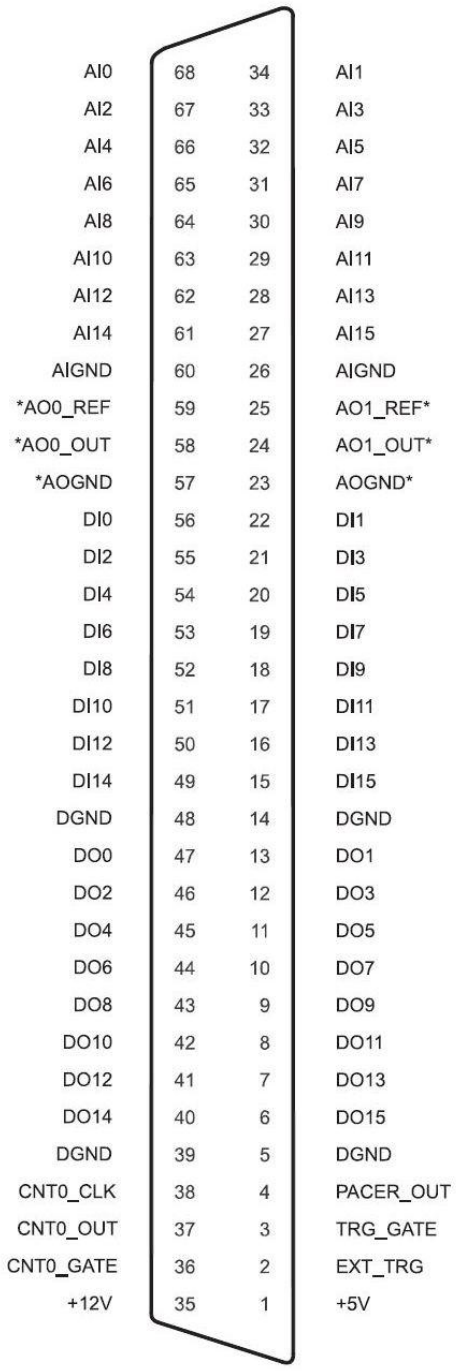

Figure 2.1: Pin Configuration PCI 1716

Table 2.2: pin configuration of PCI card

Signal Name	Reference	Direction	Description
Al<0..15>	AIGND	input	Analog Input Channels 0 through 15. Each channel pair, Al<i, +1> (i = 0, 2, 4...14), can be configured as either two single-ended inputs or one differential input.
AIGND		-	Analog Input Ground.
AOO REF A01_REF	AOGND	Input	Analog Output Channel 0/1 External Reference.
AOO OUT A01 OUT	AOGND	Output	Analog Output Channels 0/1.
AOGND			**Analog Output Ground.** The analog output voltages are referenced to these nodes.
Dl<0..15>	DGND	Input	Digital Input channels.
D0<0..15>	DGND	Output	Digital Output channels.
DGND			**Digital Ground.** This pin supplies the reference for the digital channels at the I/O connector as well as the +5VDC supply.
CNT0_CLK	DGND	Input	Counter 0 Clock Input. The clock input of counter 0 can be either external (up to 10 MHz) or internal (1 MHz), as set by software.
CNTO OUT	DGND	Output	Counter 0 Output.
CNT0_GATE	DGND	Input	Counter 0 Gate Control.
PACER_OUT	DGND	Output	Pacer Clock Output. This pin pulses once for each pacer clock when turned on. If A/D conversion is in the pacer trigger mode, users can use this signal as a synchronous signal for other applications. A low - to- high edge triggers A/D conversion to start.
TRG_GATE	DGND	Input	**A/D External Trigger Gate.** When TRG _GATE is connected to +5 V, it will enable the external trigger signal to input. When TRG GATE is connected to DGND, it will disable the external trigger signal to input.
EXT_TRG	DGND	Output	A/D External Trigger. This pin is external trigger signal input for the A/D conversion. A low - to-high edge triggers *N* D conversion to start.
+12V	DGND	Output	+12 VDC Source.
+5V	DGND	Output	+5 VDC Source.

2.4 Block Diagram

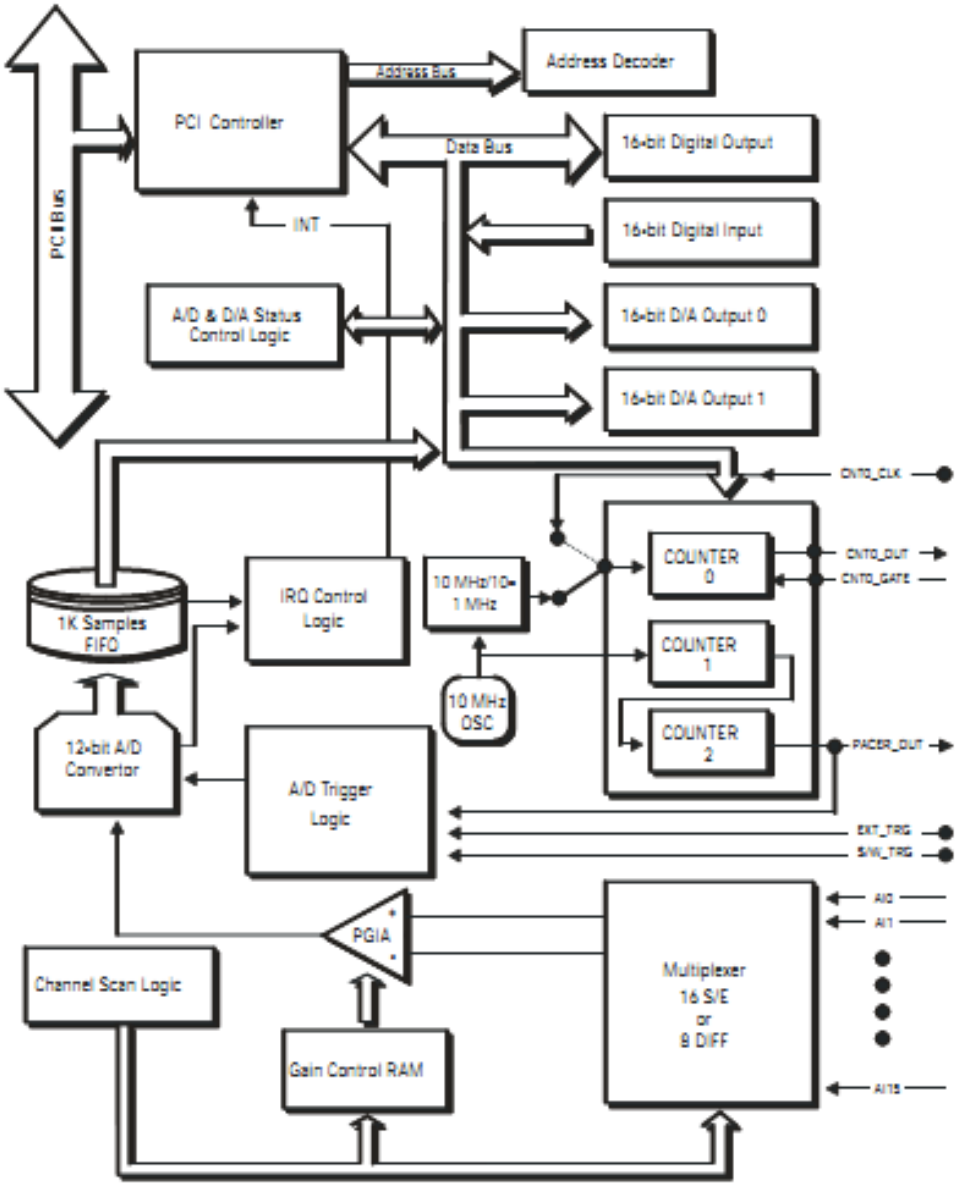

Figure 2.2: Block Diagram of PCI 1716 architecture

Chapter 3. Local Speed Control Of DC Servo Motor

For the local speed control of DC servo motor, the following steps have been followed:

1. Running the motor in open loop
2. Model estimation
3. Comparison of the open loop response of the transfer function and that of the motor
4. Designing the PID controller for the estimated model
5. Real-time execution of the system in closed loop, and comparison of the open-loop and closed-loop responses

These steps have been explained in detail, in the following sub-sections:

3.1 Running The Motor In Open Loop

(In all the following Simulink models, the configuration parameters have to be set as explained in the 5th chapter.)

When a system operates in the open loop configuration, the desired input is fed into the system, using only the current state and the model of the system. The system does not need a feedback from the output, as it does not observe the output of the process that it controls. As a result, an open loop system cannot engage in machine learning. It can neither compensate for disturbances in the system, nor engage in error-correction.

Open loop configuration is suited for systems where the relationship between the input and the resultant state can be modeled by a mathematical formula, and the load is predefined and fixed. For example, for a motor driving a constant load, the desired speed can be easily obtained in open loop by appropriately changing the input voltage. But, if

the load were unpredictable, the motor-speed would be a function of the input voltage as well as the load. In such a case, open loop control would not give satisfactory results.

Thus, an open loop system is used because of its advantages of simplicity and low cost, where the use of feedback is not critical.

The **Simulink model for running the motor in open loop** is shown below:

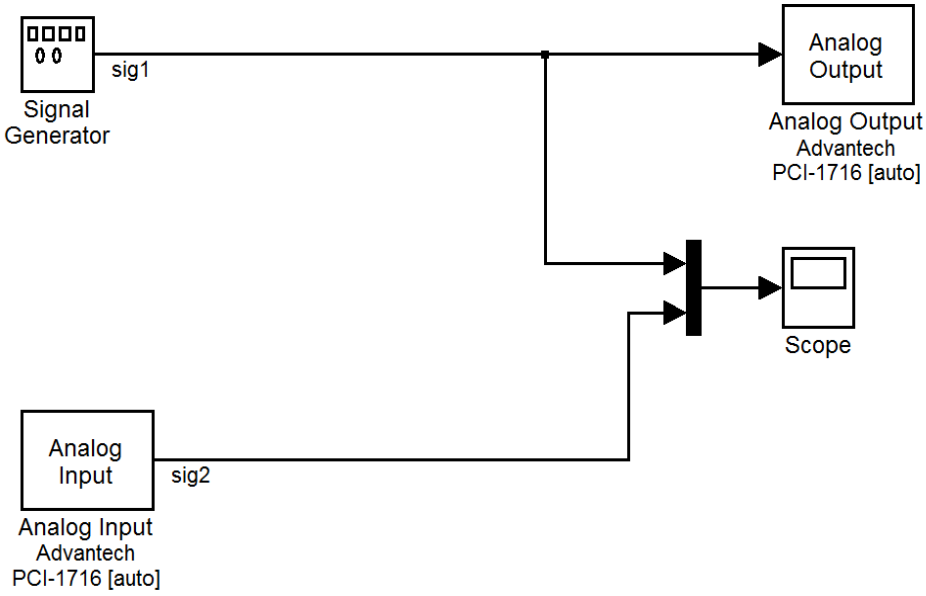

Fig 3.1: Simulink model for running the motor in open loop

Blocks Used:

1. Signal Generator
2. Analog IP/OP
3. Scope

3.1.1 Analog Input Block

Select and connect analog input channels

Library

Real-Time Windows Target

Description

The Analog Input block allows us to select and connect specific analog input channels to our Simulink model. After we add an Analog Input block to your model, we can enter the parameters for its I/O driver. The following procedure is used to configure Advantech 1716 PCI card:

- Double-click the Analog Input block. The Block Parameters: Analog Input dialog box opens:

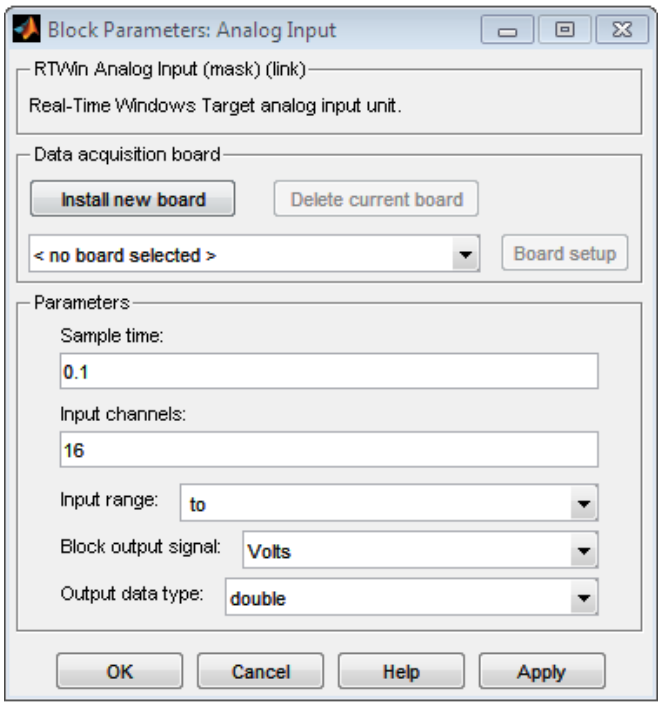

Fig 3.2: Analog Input Block Parameters Dialog Box

- In the Sample time box, enter the same value you entered in the Fixed step size box from the Configuration Parameters dialog box, or an integer multiple of that value.
- In the Input channels box, enter a channel vector that selects the analog input channels you are using on this board. The vector can be any valid MATLAB® vector form. To select all sixteen analog input channels on the Advantech 1716 board, enter

  [1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16] or [1:16]

  If you want to use the first three analog input channels, enter [1,2,3]
- From the Input range list, choose the input range for all of the analog input channels you entered in the Input channels box. The Advantech 1716 board has input range of 15V to -15V. If you want the input range to be different for different analog channels, you need to add an I/O block for each different input range.
- From the Block output signal list, choose from the following options:
  - Volts — Returns a value equal to the analog voltage.
  - Normalized bipolar — Returns a full range value of -1 to +1 regardless of the input voltage range.
  - Normalized unipolar — Returns a full range value of 0 to +1 regardless of the input voltage range. For example, an analog input range of 0 to +5 volts and -5 to +5 volts would both be converted to 0 to +1.
  - Raw — Returns a value of 0 to 2n -1. For example, a 12-bit A/D converter would return values of 0 to 212 -1 (0 to 4095). The advantage

of this method is the returned value is always an integer with no round-off errors.

- Set Output data type to specify the type of data that the block will output to the model.
- Click OK or Apply. - (MATLAB, MATLAB Help)

3.1.2 Analog Output

Select and connect analog output channels

Library

Real-Time Windows Target

Description

The Analog Output block allows us to select and connect specific analog output channels to our Simulink model. After we add an Analog Output block to our model, we can enter the parameters for its I/O driver. The following procedure is used to configure Advantech 1716 PCI card:

1. Double-click the Analog Output block. The Block Parameters: Analog Output dialog box opens:

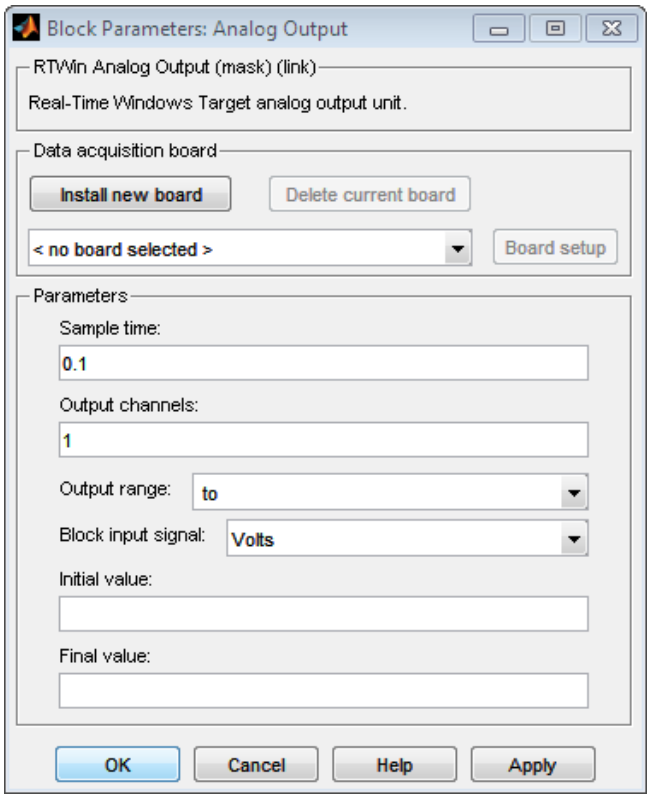

Fig 3.3: Analog Input Block Parameters Dialog Box

2. In the Sample time box, enter the same value you entered in the Fixed step size box from the Configuration Parameters dialog box, or an integer multiple of that value.
3. In the Output channels box, enter a channel vector that selects the analog output channels you are using on this board. The vector can be any valid MATLAB vector form. To select both analog output channels on the Advantech 1716 board, enter [1,2] or [1:2]
4. From the Output range list, choose the input range for all of the analog input channels you entered in the Input channels box. The Advantech 1716 board has input range of 15V to -15V. If you want the input range to be different for

different analog channels, you need to add an I/O block for each different input range.

5. From the Block input signal list, choose from the following options: Same as the Analog IP Block discussed earlier.
6. Enter the initial value for each analog output channel you entered in the Output channels box. For example, if you entered [1,2] in the Output channels box, and you want an initial value of 0 volts, enter [0,0].
7. Enter a final value for each analog channel you entered in the Output channels box. For example, if you entered [1,2] in the Output channels box, and you want final values of 0 volts, enter [0,0].
8. Click OK or Apply. - (MATLAB)

The simulation results obtained when the above model was run are shown below:

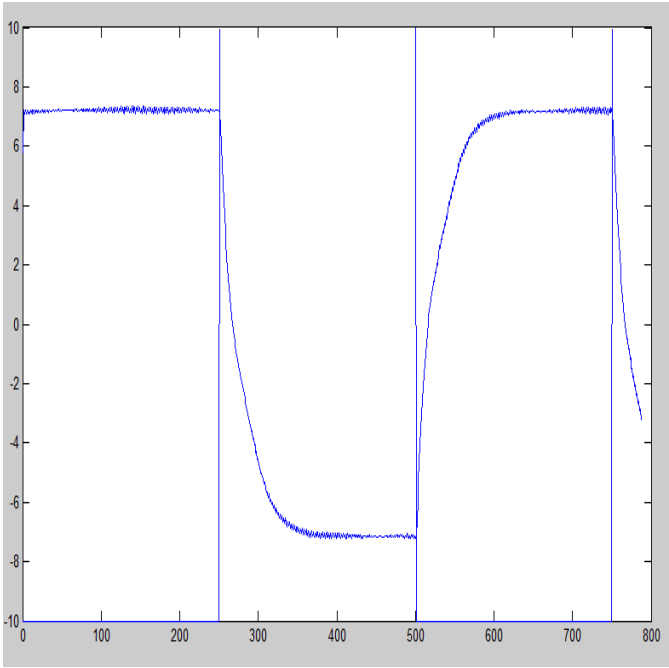

Fig 3.4: Open loop response of the motor

3.2 Model Estimation

For the purpose of any form of analysis on a plant, be it designing a controller for it, or studying its response, a convenient way is modeling the plant.

In the given case, from the open loop response of the motor, it is seen that the output of the motor in open loop, does not track the input. So, the motor needs to be run in closed loop configuration. To design the controller for closed loop operation, first, modeling of the plant has been done, as explained in detail, in the following sub-sections.

Transfer function modeling of the plant is one of the methods. A transfer function (also known as the system function or network function) is a mathematical representation, in terms of spatial or temporal frequency, of the relation between the input and output of a linear time-invariant system.

The transfer function that relates the input voltage and the speed of the DC servo motor is a second order one, as already proved in section 1. This transfer function can be estimated in MATLAB using the following procedure:

The System Identification Toolbox comes in handy for this purpose. The whole progress of the transfer function identification is represented by a system identification tool session.

Step-1: Importing relevant data from the model to MATLAB workspace:

"To Workspace" blocks are used in the Simulink model of the plant, for importing the following to MATLAB workspace:

a) The voltage input from the signal generator and
b) The speed output from the taco-generator of the plant (which is obtained at the Analog Input block of the model)

This has been shown in the Simulink model given below:

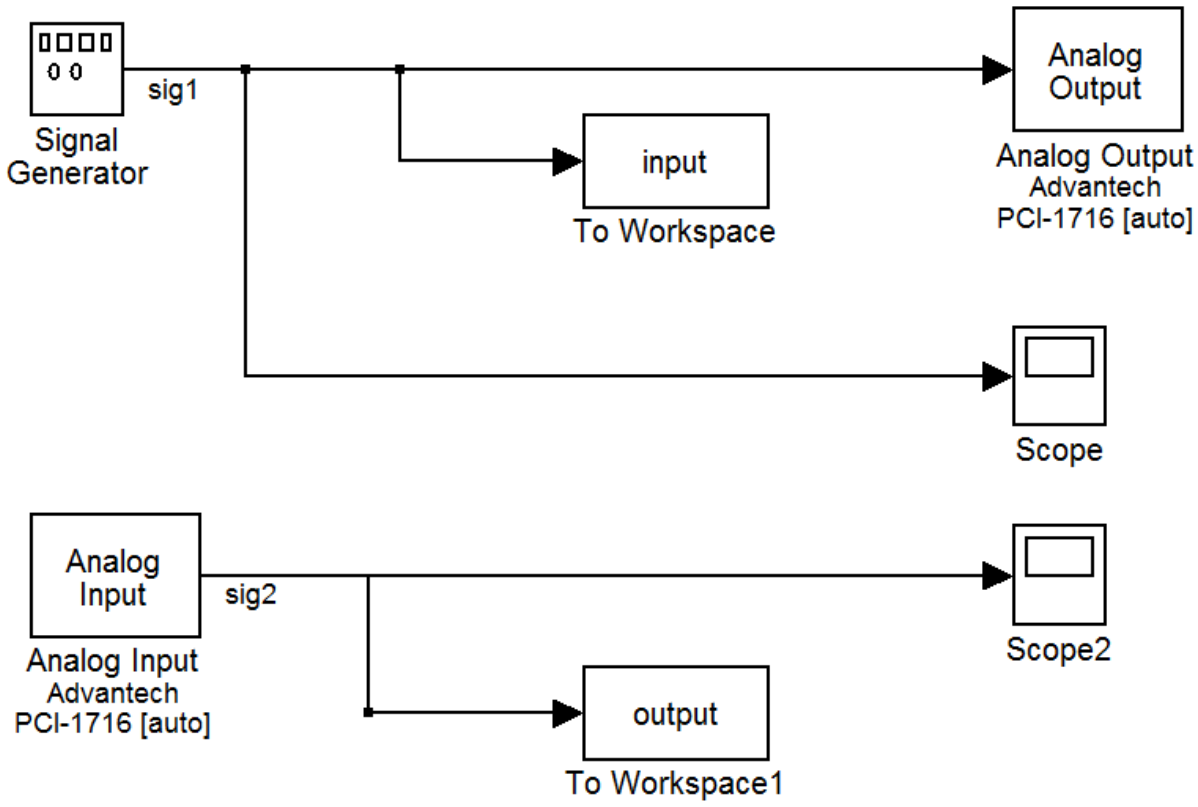

Fig 3.5: Simulink Model to get the system response.

The analog input and output blocks are set with values as shown previously.

The model is then run, to import the voltage input and speed output to the workspace.

Step-2: Starting a new session in the System Identification Tool GUI, or opening a saved session:

A new session can be started by either typing “ident” in the MATLAB command window, or by selecting Start -> Toolboxes -> System Identification -> System Identification Toolbox GUI in the MATLAB Desktop:

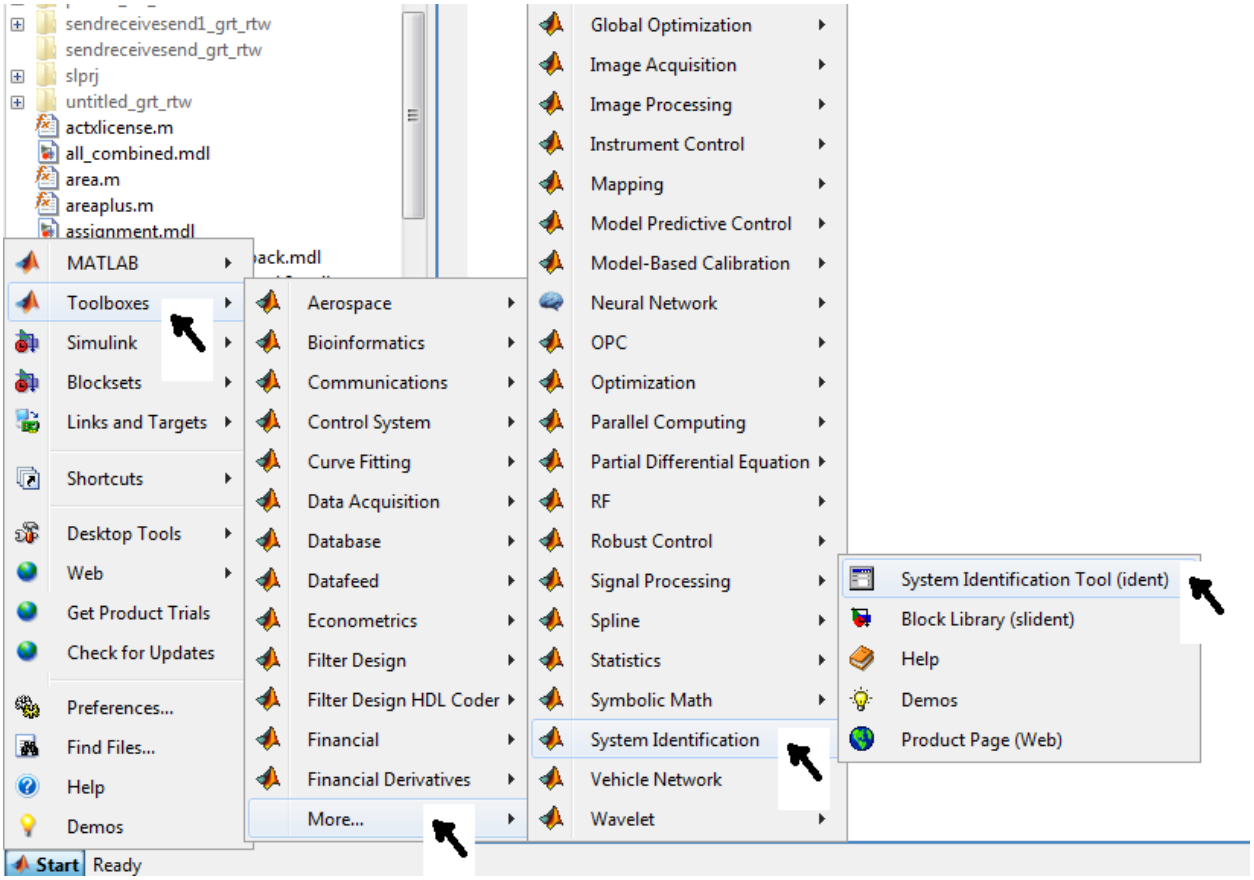

Fig 3.6: System Identification Tool

The System Identification Tool GUI opens in a new window, as shown below:

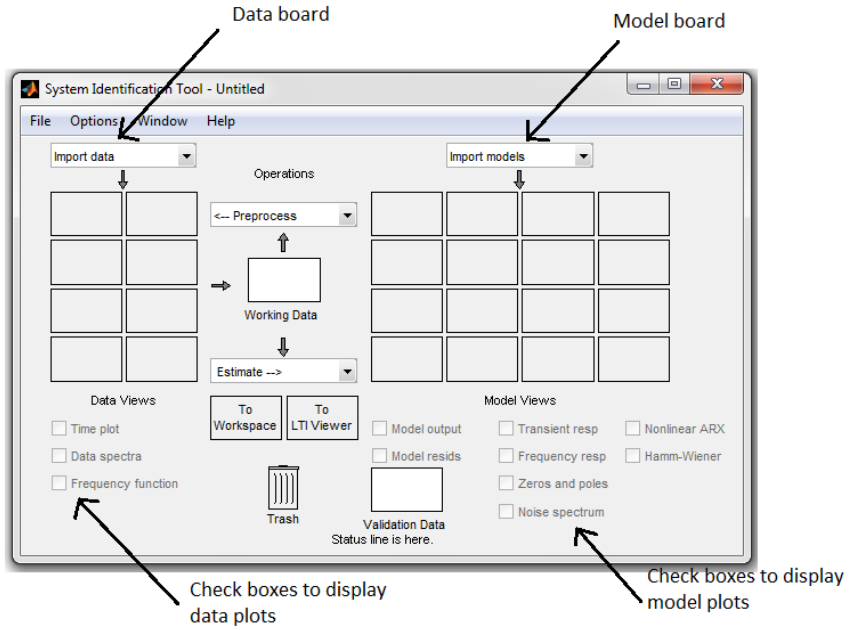

Fig 3.7: System Identification Tool Dialog Box

The data board contains rectangular icons which show the data imported into the GUI. The model board contains icons which represent the models estimated or imported into the GUI. Model icons can be dragged and dropped in the model board into open dialog boxes.

The current session may be closed by selecting File -> Close Session. This prompts to save the current session, if it has not already been saved.

Step-3: In the "Import Data" dropdown menu, select "Time domain data". The following window opens, where the name of the input and output To Workspace blocks have been entered.

Also, depending on the starting time of the data in the data acquiesced, the starting time is entered; and depending on the sampling time set in the model of the plant, the sampling interval is entered.

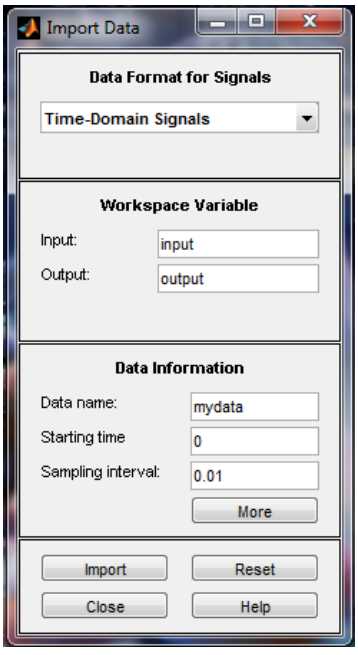

Fig 3.8: Import Data Dialog Box

Step-4:

Click on "Import". The following window opens, where the input and output data from the model, have been imported into the data board by the name of "mydata":

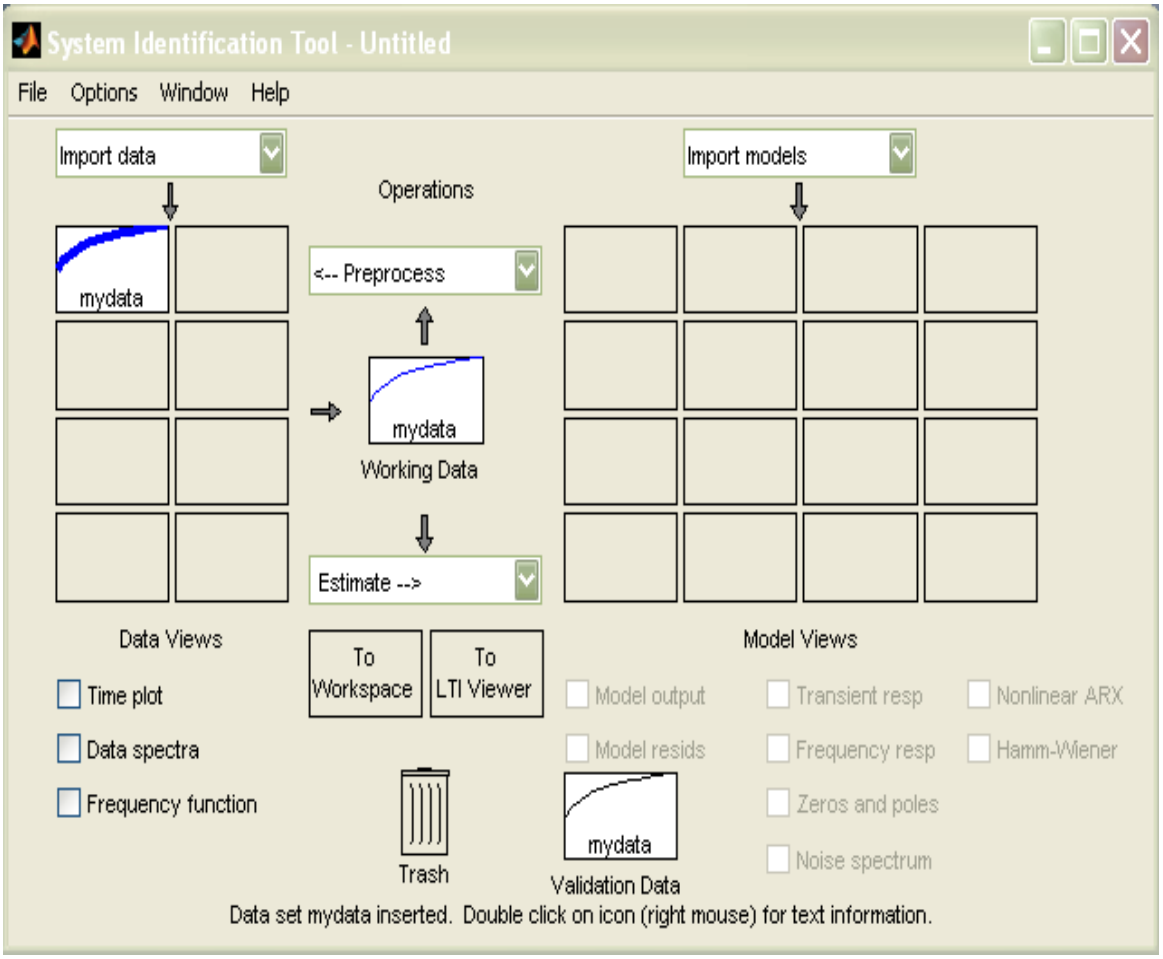

Fig 3.9: System Identification Tool Dialog Box

Step-5:

In the Estimate drop-down menu, click on Process Models. The following window opens, where the model has been chosen to have 1 zero and 2 poles, but no delay and integrator:

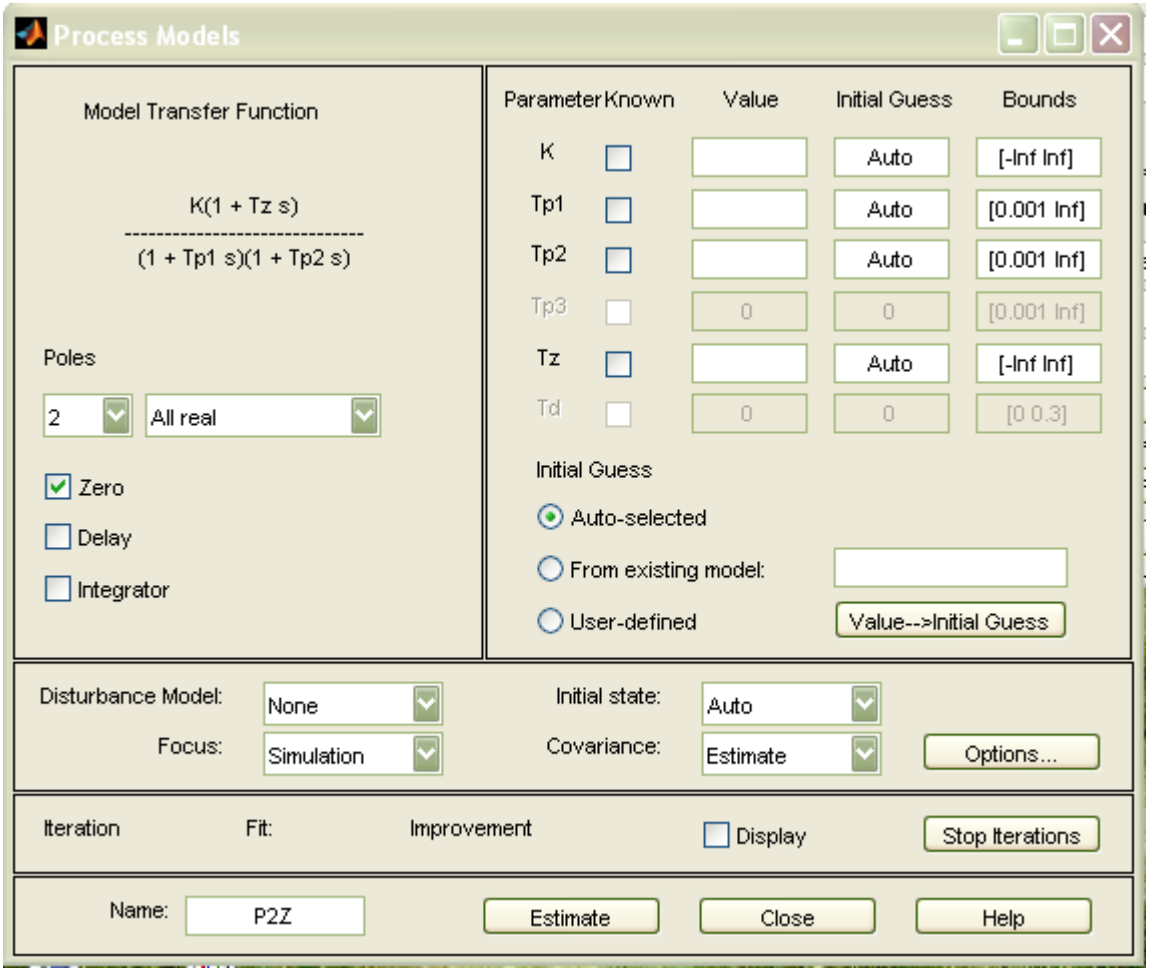

Fig 3.10: Process Models Dialog Box

The transfer function of the model, in abstract form, is also given in the window on the top left corner.

Step-6:

Click on Estimate. The following window opens.

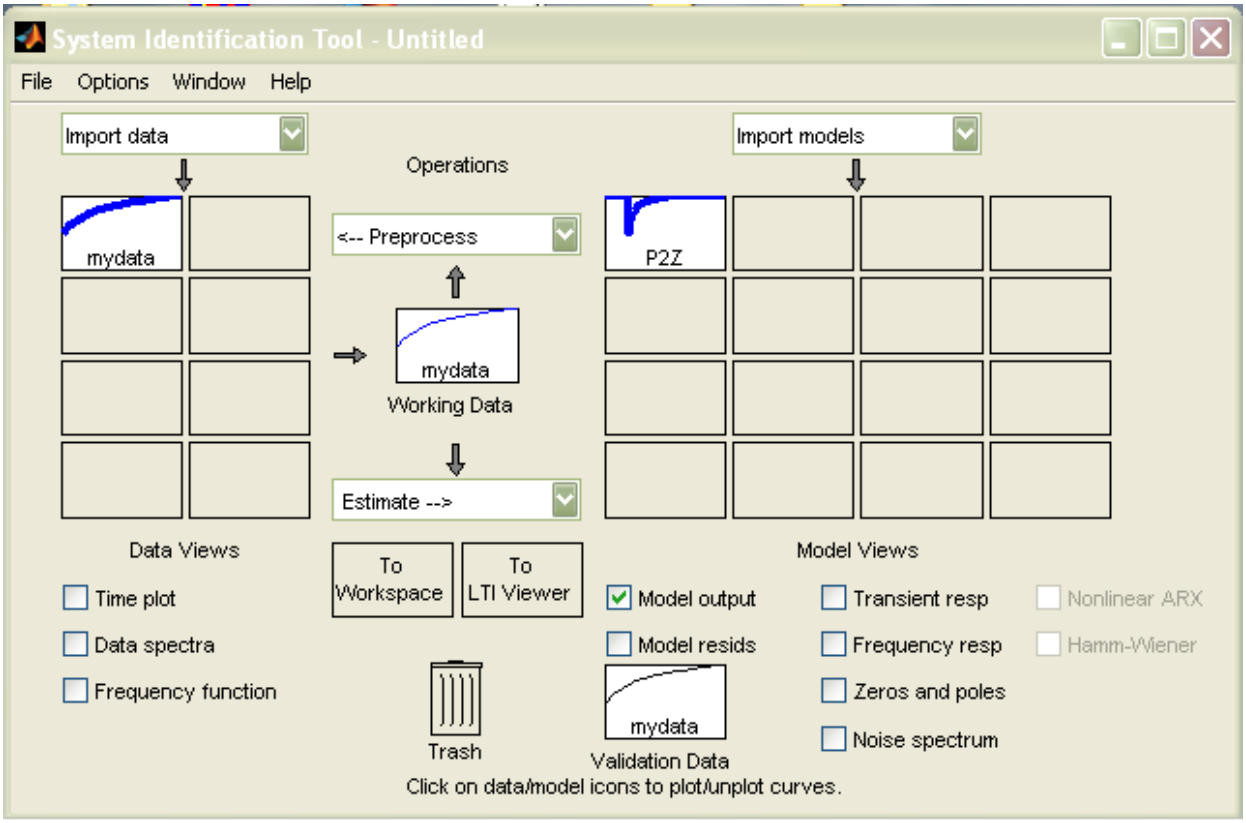

Fig 3.11: System Identification Tool Dialog Box

Step-7:

As shown in the above window, check Model Output. The following estimate of the plant, is obtained in a new window:

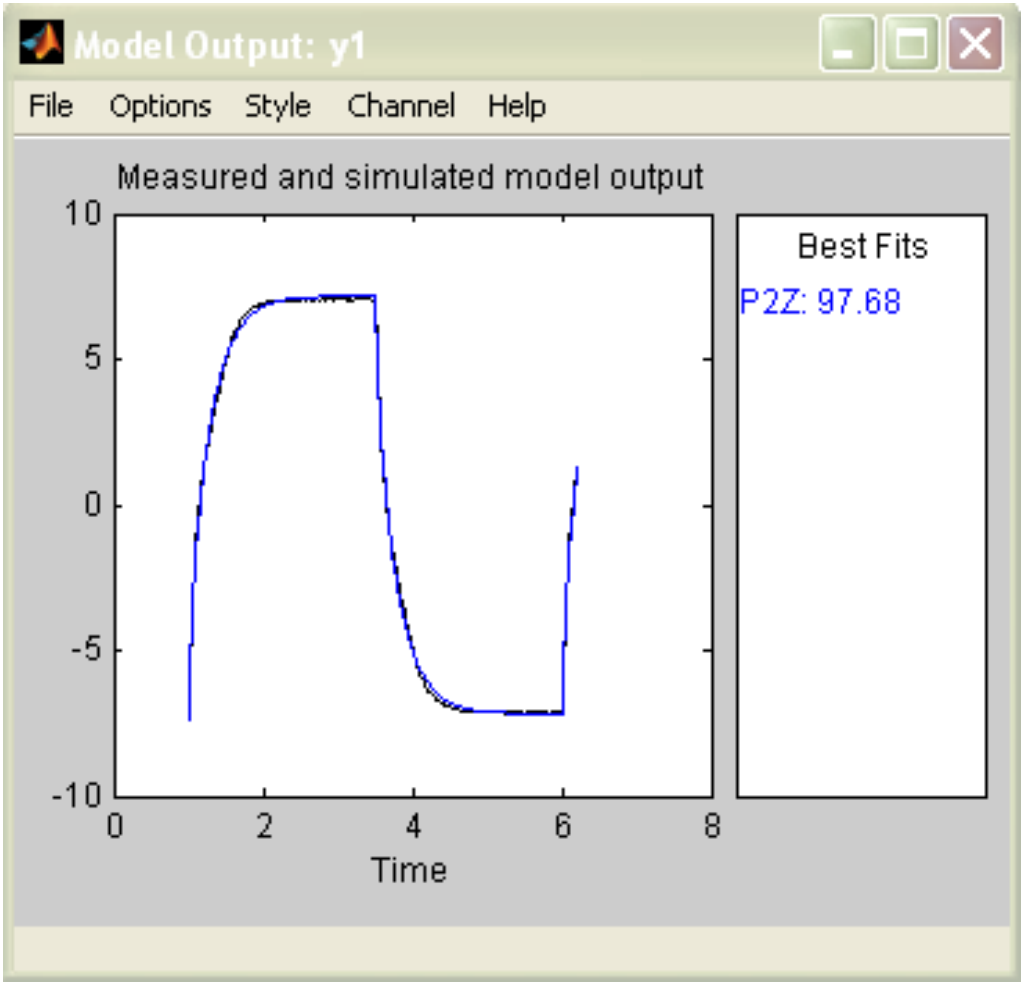

Fig 3.12: Model Output Dialog Box

With the chosen configuration of 1 zero and 2 poles, and no delay or integrator, as shown above, a best fit of 97.68% is obtained. This is a satisfactory estimation.

If the fit had been less than satisfactory, then changing of the configurations would have been required. Once, a satisfactory fit is obtained, the next and final step is followed.

Step-8:

The model which gives the most satisfactory fit, is dragged and dropped from the model board into the To Workspace area. This moves the values of Kp, Tz, Tp1 and Tp2 to the workspace, from which, the corresponding transfer function is estimated.

For the fit of 97.68% shown above, the transfer function parameters are:

Kp = 0.74153

Tp1 = 0.30208

Tp2 = 0.0012424

Tz = 0.039167

And the corresponding transfer function is:

$$\frac{0.02904\,s + 0.7415}{0.0003753\,s^2 + 0.3033\,s + 1}$$

3.3 Comparison Of The Open Loop Response Of The Transfer Function And That Of The Motor

The Simulink block diagram for comparison of open-loop response of the transfer function and that of the motor is shown on the next page.

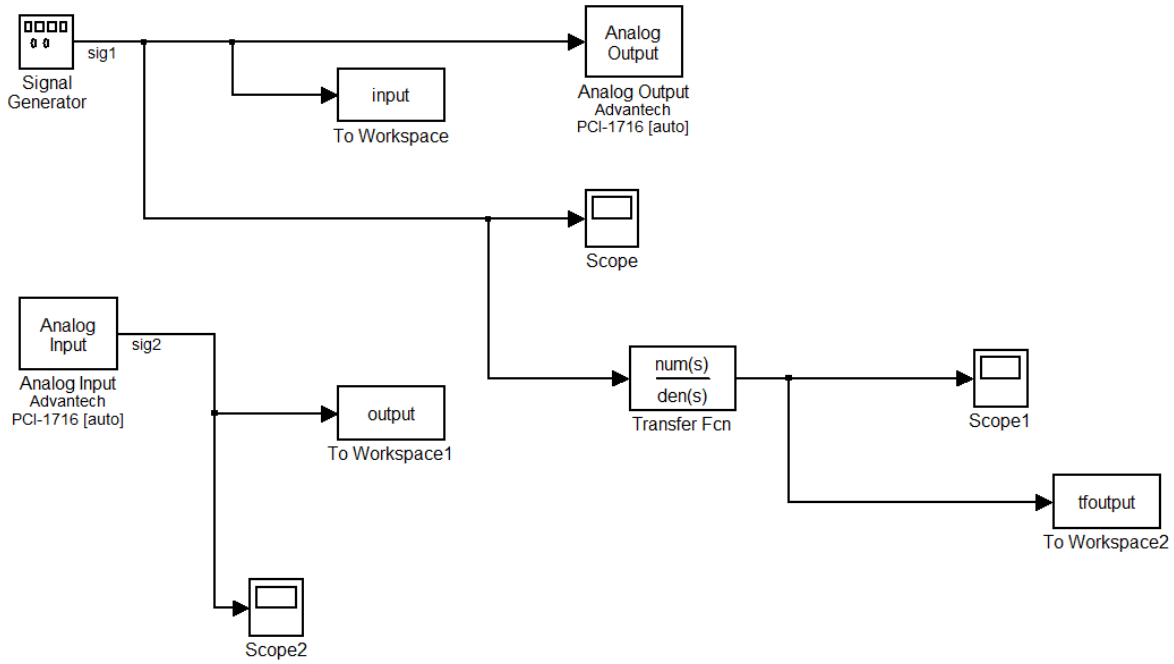

Fig 3.13: Simulink model for comparison of open-loop response of the transfer function and that of the motor

The analog input and output blocks are se with values as shown previously. And the transfer function block Is set with the transfer function obtained in the previous sub-section.

The simulation results are as shown below:

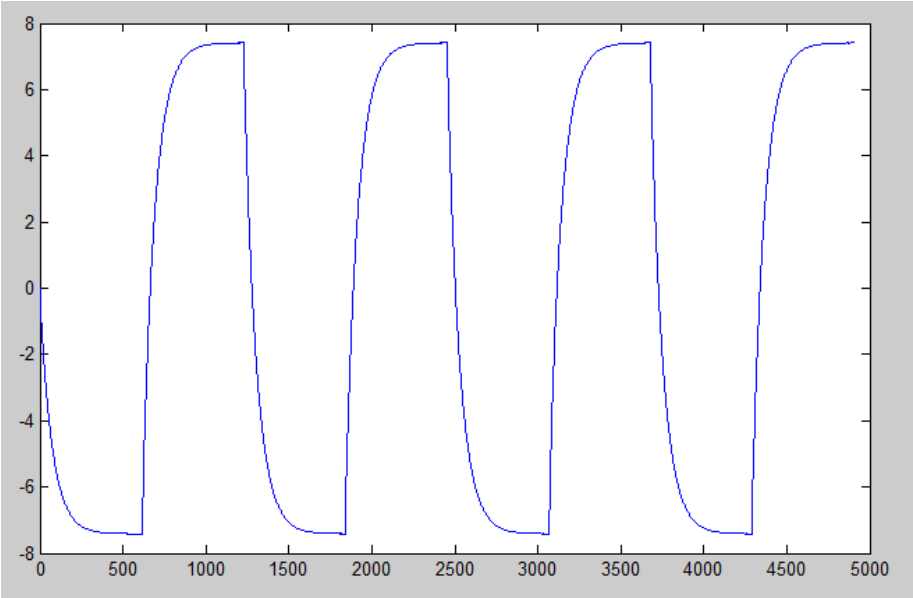

Fig 3.14: Transfer function output

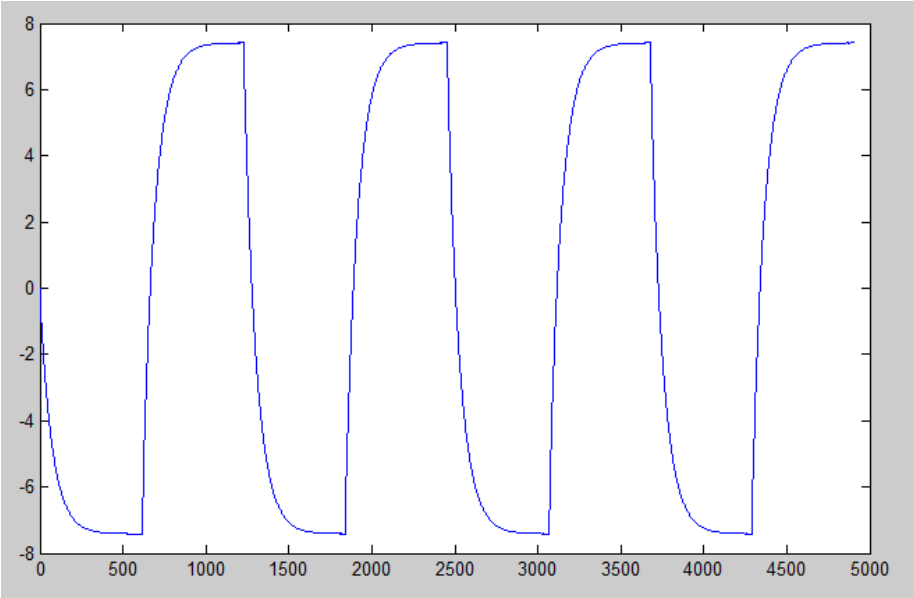

Fig 3.15: Plant output

From these results, it is seen that the transfer function's open loop response is more or less, same as the open loop response of the motor. But, neither open loop response successfully tracks the input (or the reference) which is set as a square wave varying between -10 volts and +10 volts. This indicates the necessity of a closed loop control

system for the motor, so that the transfer function (and the plant) output successfully tracks the reference.

This is achieved with the help of a PID controller, which has been described in the next sub-section.

3.4 Designing The PID Controller For The Estimated Model

The PID controller in a closed loop control system is shown below:

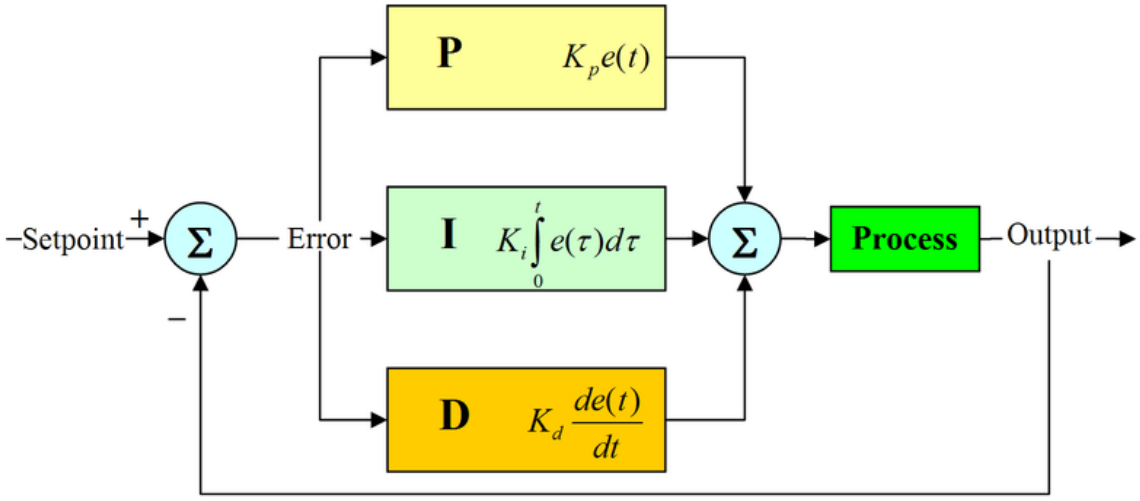

Fig 3.16: PID controller in closed loop

A proportional–integral–derivative controller (PID controller) is a controller which is popularly used in industrial control systems . It is fed with the error signal, that is, the difference between the reference, or the desired output and the actual output (which is obtained as a feedback). The controller then attempts to bring the actual output to track the reference.

The PID controller algorithm involves three separate constant parameters (proportional, integral and derivative, denoted by P, I and D respectively) and is thus, also called three-term control. P depends on the present error, I on the accumulation of past errors, and D

is a prediction of future errors, based on current rate of change. The weighted sum of these three actions is given as input to the process through a control element. By tuning the three parameters, the controller provides the required control action.

If u(t) represents the controller output, which is the manipulated variable (MV(t)), that is, the process input, then:

$$u(t) = MV(t) = K_p e(t) + K_i \int_0^t e(\tau)d\tau + K_d \frac{d}{dt} e(t)$$

where

P_{out}: Proportional term of output

K_p: Proportional gain, a tuning parameter

K_i: Integral gain, a tuning parameter

K_d: Derivative gain, a tuning parameter

e: Error = SP − PV

t: Time or instantaneous time (the present)

Effect of proportional gain: If the proportional gain is too small, it results in a small output response for a large input error. This, results in a large steady-state error. In a pure proportional controller, this steady state error is retained, and is called “Droop”. Droop is proportional to the process gain and inversely proportional to the proportional gain. ($e = G / Kp$) Droop may be compensated for, by adding a bias term that is, setting the set-point as more than the actual reference value; or by adding an integral term. However, if the proportional gain is too high, it may result in system-instability.

Effect of integral gain: The integral gain removes the residual steady state error that occurs with a pure proportional controller. However, a very high integral gain results in overshoot.

Effect of derivative gain: The derivative gain is used to reduce the overshoot caused by the integral gain and improve the combined controller-process stability. But, it slows down the transient response of the system as well as, increases sensitivity of the system to noise. So, a lead-compensator is used as an approximation of the differentiator.

3.4.1 PID tuning:

Tuning a control loop refers to adjusting its control parameters, that is, the proportional, integral and derivative gains, so as to obtain the desired system response.

For systems having a degree of non-linearity, gains that work out well at full load conditions, give erroneous results at starting or light load conditions. To solve this problem, gain scheduling may be used. It uses different gains at different operating regions.

Default tuning might give the desired results in some cases, but in others, careful tuning of the PID is required. To deal with the difficult problem of PID tuning (so that the gains satisfy complex criteria within the limitations of PID control), various methods of PID control are available.

Of these methods, relay-oscillation method (Ziegler-Nichols method) and Pole Placement method have been used for the design of the PID controller for speed control of DC servo motor. These methods have been explained below:

1. Relay oscillations method:

   The following Simulink model is used, for this method.

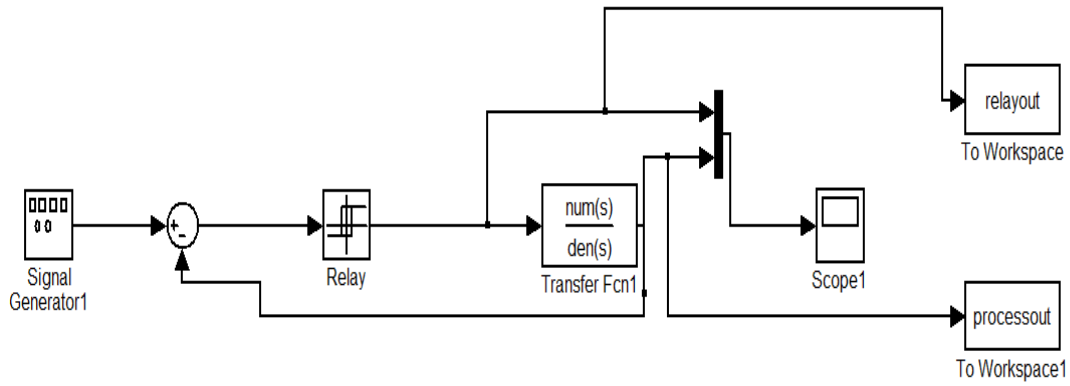

Fig 3.17: Relay oscillation Simulink model

This is similar to bang-bang control, where whatever the response, the relay block pushes it in the opposite direction. This gives automatic oscillations of the process output. (This bypasses the limitation imposed by the method where the proportional gain is to be increased till for its minimum value, the relay output starts to oscillate. So, it is difficult to obtain the range of the proportional gain, which is different for each process.)

The scope output showing the relay output and the process output is as shown below:

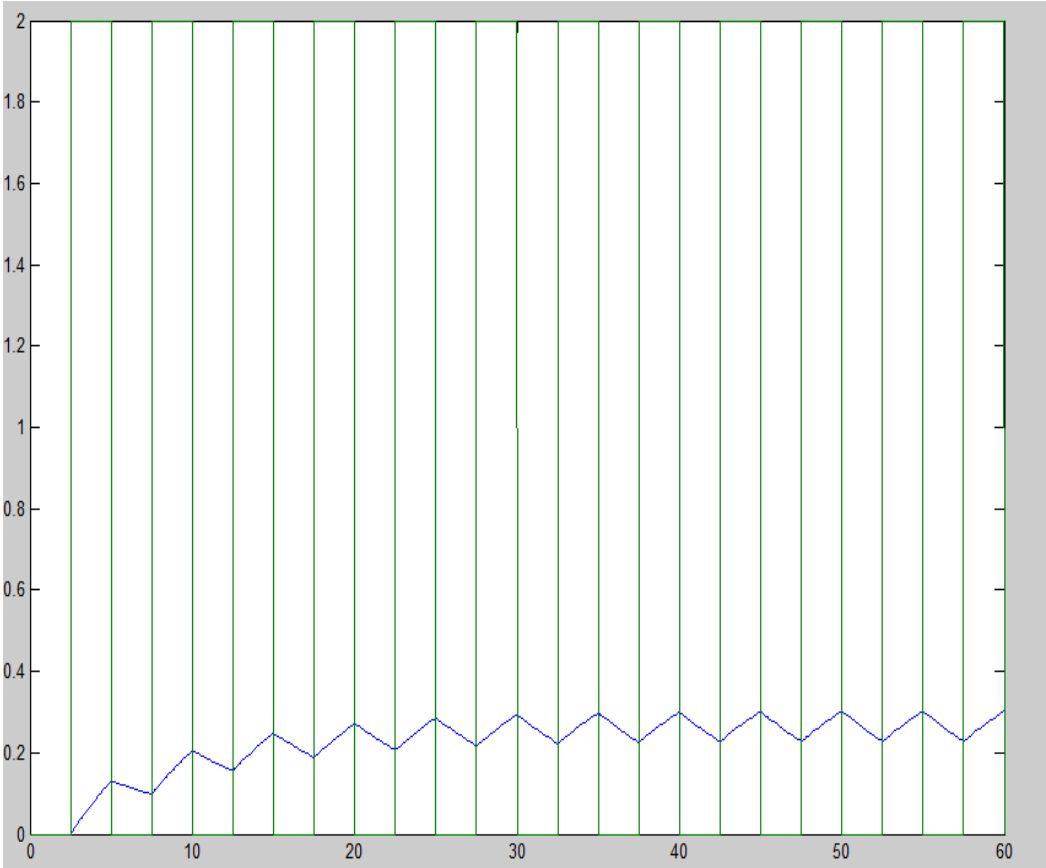

Fig 3.18: Scope output showing the relay output and the process output

Say, the relay output (shown in green) oscillates with an amplitude of'd', and the process output (shown in blue) oscillates with an amplitude of 'a'.

Then, the ultimate gain is given by:

$$Ku = \frac{(4 \times d)}{(pi \times a)}$$

And the ultimate time-period T_u = Time period of oscillation of process output

Then, the PID gains are given by:

$$Kp = 0.6 \times Ku$$

$$Ti = 0.4 \times Tu$$

$$Td = 0.12 \times Tu$$

From the graph shown above,

$$d = 1$$

$$a = 0.741$$

Then, $Ku = \frac{(4 \times 1)}{(pi \times 0.741)} = 1.718$

And $Tu = 5$ (measured from the graph)

So, the PID gains obtained by the relay – oscillations method are:

$$Kp = 0.6 \times 1.718 = 1.03$$

$$Ti = 0.4 \times 5 = 2$$

$$Td = 0.12 \times 5 = 0.6$$

2. Pole-placement method:

The block-diagram for the closed loop system, using PID controller is as given below:

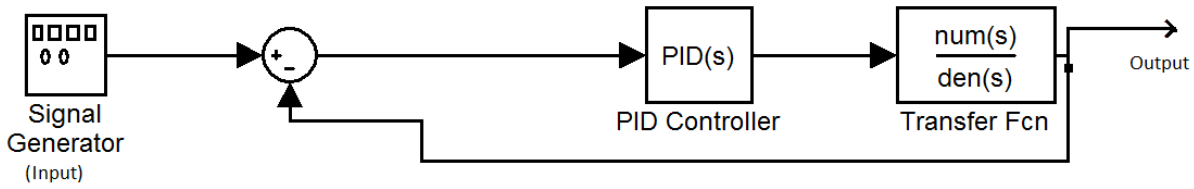

Fig 3.19: Closed loop PID controller

For this system, the open loop transfer function is given by:

$$G = G(pid) \times G(motor)$$

Where $G(pid) = Kp + \left(\frac{Ki}{s}\right) + (Kd \times s)$

$$= \frac{K_d s^2 + K_p s + K_i}{s}$$

And $G(motor) = \frac{0.029 \times s + 0.74}{0.004 \times s^2 + 0.3 \times s + 1}$

So,

$$G = \frac{(K_d \times s^2 + K_p \times s + K_i)(0.03 \times s + 0.74)}{0.004 \times s^2 + 0.3 \times s + 1}$$

$$= \frac{[0.03 \times K_d \times s^3 + (0.03 \times K_p + 0.74 \times K_d) \times s^2 + (0.03 \times K_i + 0.74 \times K_p) \times s + 0.74 \times K_i]}{0.004 \times s^2 + 0.3 \times s + 1}$$

And the feedback transfer function H = 1

So, the characteristic equation in terms of PID gains is given by:

$$1 + G \times H = 0$$

$$0.004 \times s^2 + 0.3 \times s + 1 + 0.03 \times K_d \times s^3 + (0.03 \times K_p + 0.74 \times K_d) \times s^2 + (0.03 \times K_i + 0.74 \times K_p) \times s + 0.74 \times K_i = 0$$

$$(0.03 \times K_d) \times s^3 + (0.03 \times K_p + 0.74 \times K_d + 0.004) \times s^2 + (0.3 + 0.03 \times K_i + 0.74 \times K_p) \times s + (0.74 \times K_i) = 0$$

…….. (1)

Say, the desired poles are (-3+j) and (-3-j). But, since the characteristic equation needs to be a cubic equation, so another pole needs to be chosen such that it does not affect the speed of the response, and also does not bring about much change in the overall system-response. For this, it needs to be located at an optimally large distance from the dominant poles. Say, the third pole is at (-10).

So, the desired characteristic equation is:

$$(s + 10) \times (s + 3 + j) \times (s + 3 - j) = 0$$

$$\Rightarrow (s + 10) \times (s^2 + 9 + 6 \times s + 1) = 0$$

$$\Rightarrow (s + 10) \times (s^2 + 6 \times s + 10) = 0$$

$$\Rightarrow s^3 + 10 \times s^2 + 6 \times s^2 + 60 \times s + 10 \times s + 100 = 0$$

$$\Rightarrow s^3 + 16 \times s^2 + 70 \times s + 100 = 0$$ …….. (2)

Comparing the characteristic equation in terms of PID gains (1) and the desired characteristic equation (2),

$$0.03 \times K_p + 0 \times K_i + 0.26 \times K_d = -0.004 \ldots\ldots (3)$$

$$0.74 \times K_p + 0.03 \times K_i - 2.1 \times K_d = -0.3 \ldots\ldots (4)$$

$$0 \times K_p + 0.74 \times K_i - 3 \times K_d = 0 \ldots\ldots\ldots\ldots (5)$$

Solving equations (3), (4) and (5):

$Kp = 0.34$

$Ki = 0.097$ and

$Kd = 0.024$

Using the PID gains obtained above, and then, fine-tuning, the PID gains are obtained as:

$Kp = 4$

$Ki = 2$ and

$Kd = 0.1$

These PID gains are then entered, in the following Simulink model:

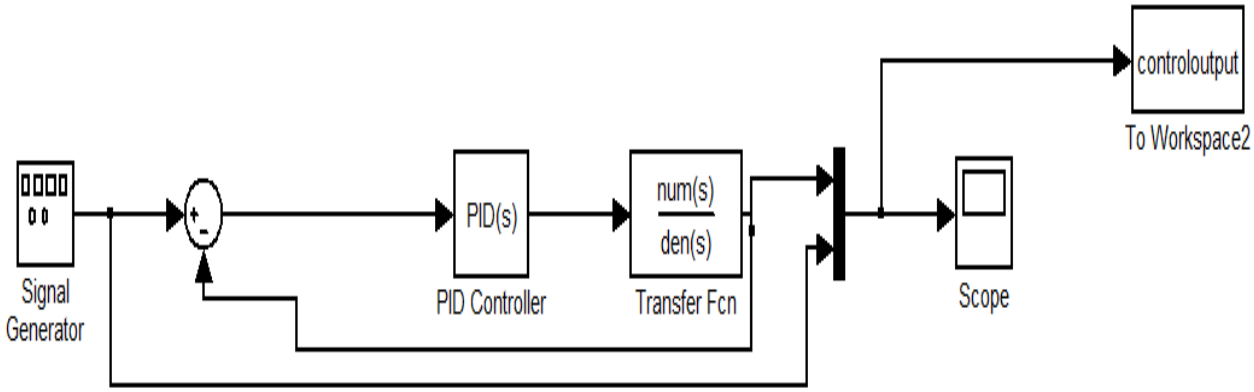

Fig 3.20: Closed loop PID controller

When this model is run, the output of the closed loop system is seen to closely follow the input from the signal generator. This indicates optimum performance of the system.

The signal generator's input, as well as, the process output is shown in the simulation result, shown on the next page:

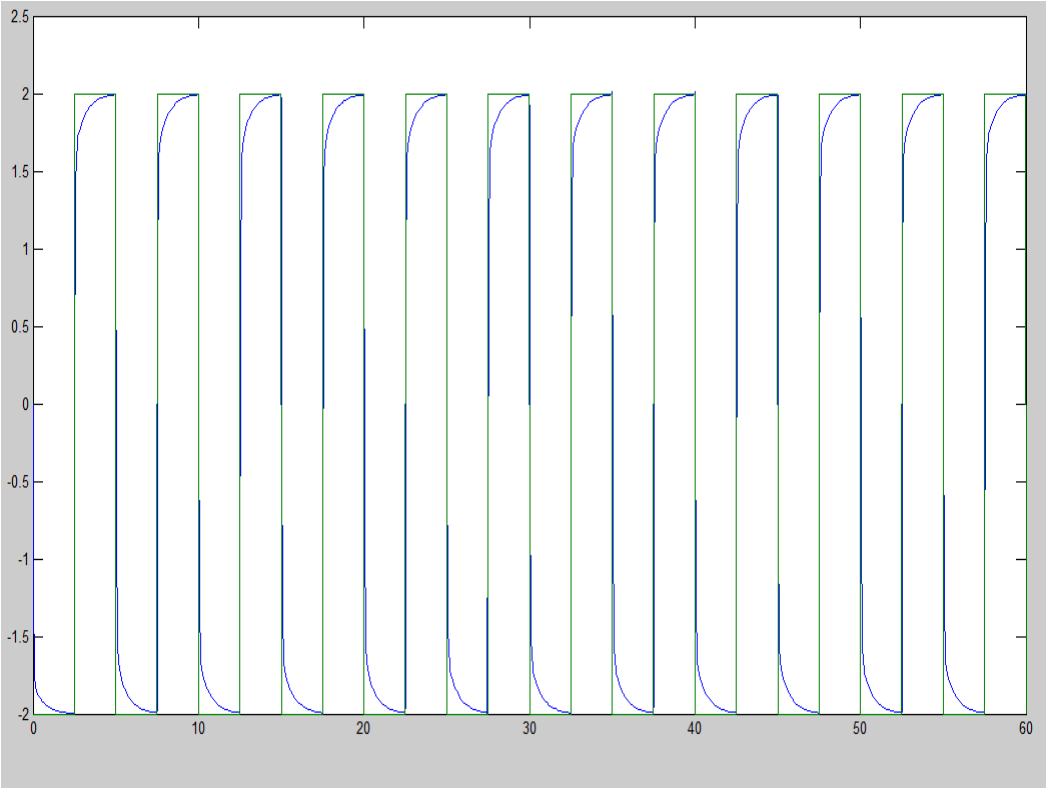

Fig 3.21: TF Output

Here, the input from the signal generator is shown in green, and the output of the closed loop control system is shown in blue.

3.5 Real Time Execution Of The System In Closed Loop, And Comparison Of The Open Loop And Closed Loop Responses

The closed loop control system for the motor using a PID controller is modeled as the Simulink model shown below. Here, the analog input and output blocks are set as shown previously. The saturation block is used to limit the PID output to certain prefixed values.

The PID block parameters are also shown below the model:

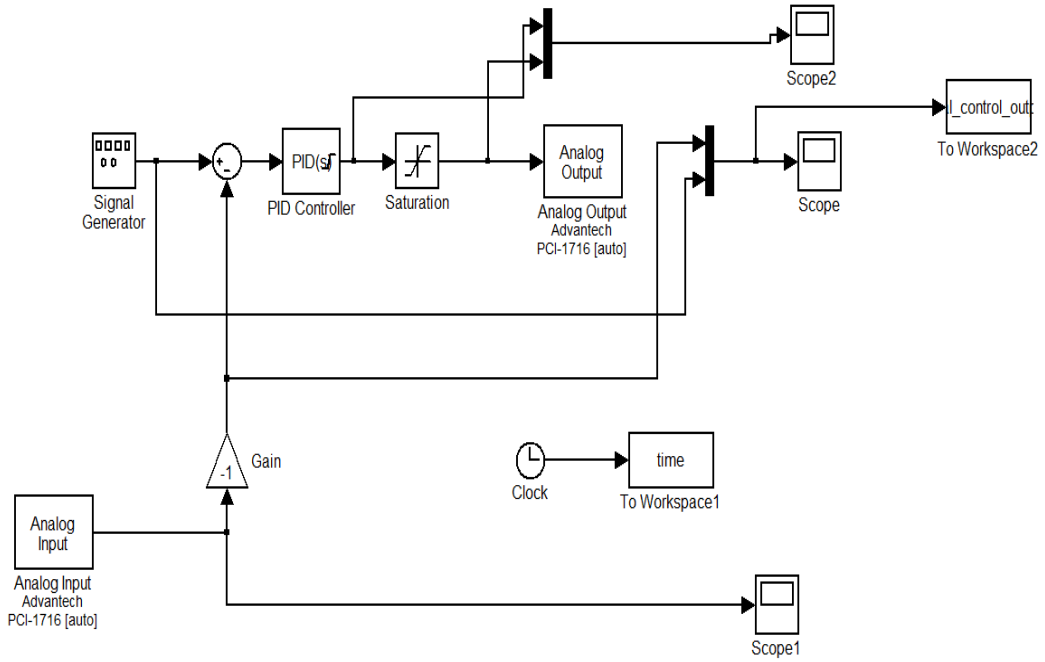

Fig 3.22: PID Control Model of real system

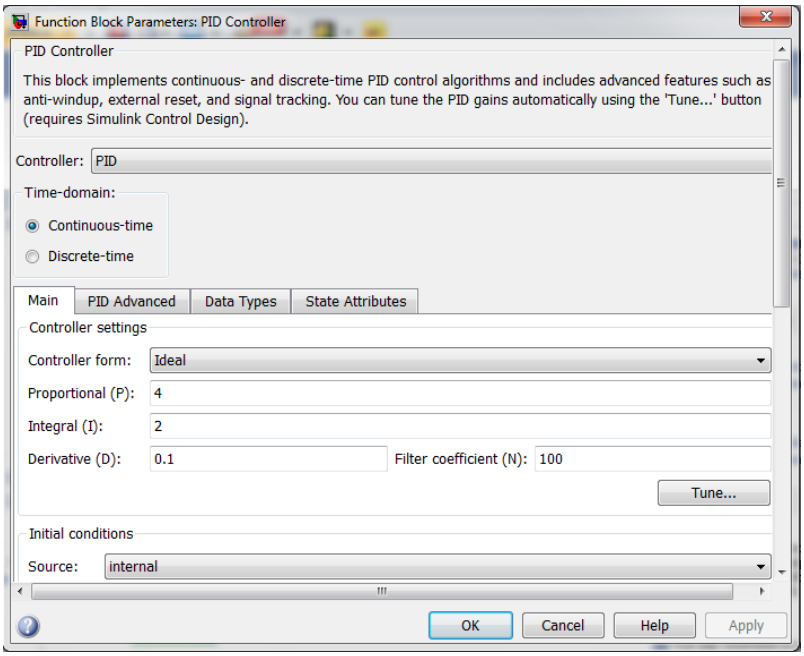

Fig 3.23: PID Block Parameters Dialog Box

The simulation result obtained on running the above model is as shown below:

Simulation result comparing the input and the output of the closed loop control system (controlled by a tuned PID controller):

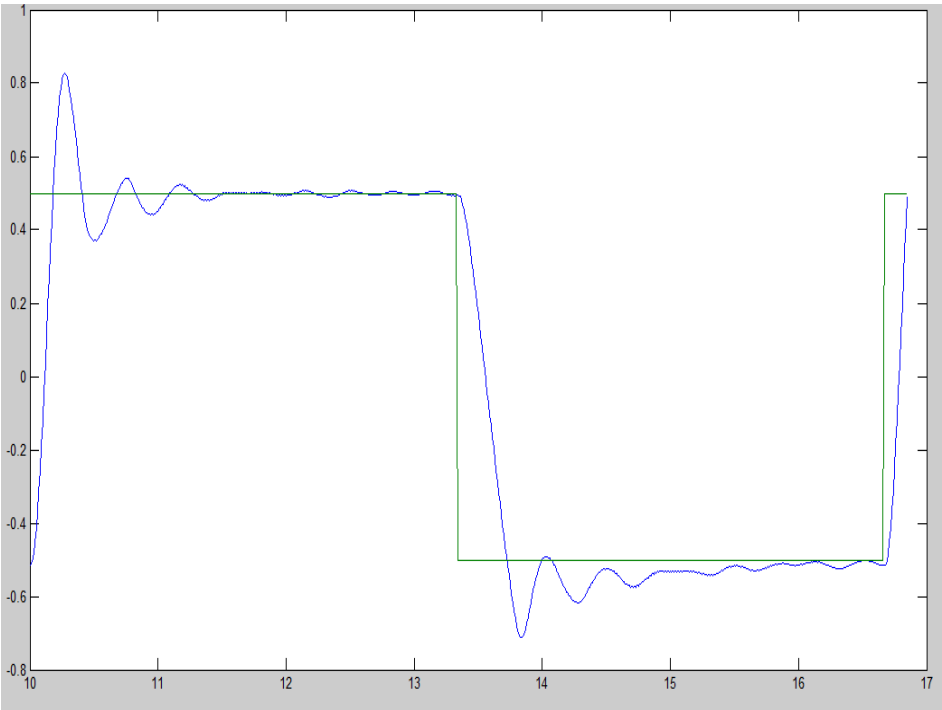

Fig 3.24: Input / Output Graph

From the above closed loop response of the system, it is seen that the output tracks the input. So, the purpose of speed control of the DC servo motor has been successfully achieved.

Chapter 4. What Is UDP?

The User Datagram Protocol (UDP) is one of the core members of the Internet Protocol Suite, the set of network protocols used for the Internet. With UDP, computer applications can send messages, in this case referred to as datagrams, to other hosts on an Internet Protocol (IP) network without requiring prior communications to set up special transmission channels or data paths. The protocol was designed by David P. Reed in 1980 and formally defined in RFC 768. - (Wikipedia)

4.1 Why use UDP?

UDP uses a simple transmission model without implicit handshaking dialogues for providing reliability, ordering, or data integrity. Thus, UDP provides an unreliable service and datagrams may arrive out of order, appear duplicated, or go missing without notice. UDP assumes that error checking and correction is either not necessary or performed in the application, avoiding the overhead of such processing at the network interface level. Time-sensitive applications often use UDP because dropping packets is preferable to waiting for delayed packets, which may not be an option in a real-time system. - (Wikipedia)

Packet Output Block

Transmit unformatted binary data

Library

Real-Time Windows Target

Description

The Packet Output block sends unformatted binary data. After you have added a Packet Output block to your model, double-click the Digital Output block to open the Block Parameters:

The driver must be Standard Devices > Serial Port, Standard Devices > UDP Protocol, Standard Devices > File, or Vector > CAN Device. Specify parameter values as needed, then click OK or Apply. When you install a UDP device, enter port addresses in decimal format in the Standard Devices UDP Protocol dialog box.

The parameters are:

Sample time

Enter the same value you entered in the Fixed step size box from the Configuration Parameters dialog box, or an integer multiple of that value.

Packet identifier

Enter the ID of the packet to receive. If your protocol does not have packet IDs, this parameter is disabled.

Output packet size

It is the number of bytes to be transmitted in the output packet. This number must be the same as the number of bytes needed to satisfy the type specifications in Output packet field data types.

Output packet field data types

A string, or a cell array of strings, that specify how data provided by the application will be formatted into a packet for output to the device. The Packet Output block has an input port corresponding to each string in Output packet field data types. Changing the number of strings automatically changes the number of ports.

Each string has the format [n*]datatype. The data described by the string has the type specified by data-type and the width specified by n; or 1 if n is not specified. For example, 'double' means one double value, and '4*int8' means a vector of four int8 values.

The signal input to each port of the Packet Output block can be a scalar or vector of any Simulink data type. The string for each port specifies the type to be used when its signal is output to the device. If the format string for a port matches the type of the signal input to that port, the signal value appears verbatim in the output packet.

You can also perform type conversion on output. For example, if an input signal is a four-element int16 vector, but the corresponding string is '4*int8', each of the four integers is converted to an int8 before being written to the packet. The resulting data occupies four bytes in the output packet.

Show "Data Ready" port

If enabled, the block has an output port that signals 1 if the block is ready to accept new data, and 0 otherwise.

Show "Data Error" port

If enabled, the block has an output port that signals 1 if a data error has occurred, and 0 otherwise.

Initial value

If specified, a vector that has the same number of elements as the sum of the widths of the input signals across all ports. The specified data is sent when simulation begins, before any other data that is output during simulation.

Final value

If specified, a vector that has the same number of elements as the sum of the widths of the input signals across all ports. The specified data is sent when simulation ends, after any other data that is output during simulation. - (MATLAB, MATLAB Help)

Packet Output block has similar settings.

Chapter 5. Set up of UDP communication between two computers using MATLAB.

5.1 Steps for the Local PC.

Step 1. Install **real time kernel** in MATLAB.

Command : rtwintgt –install

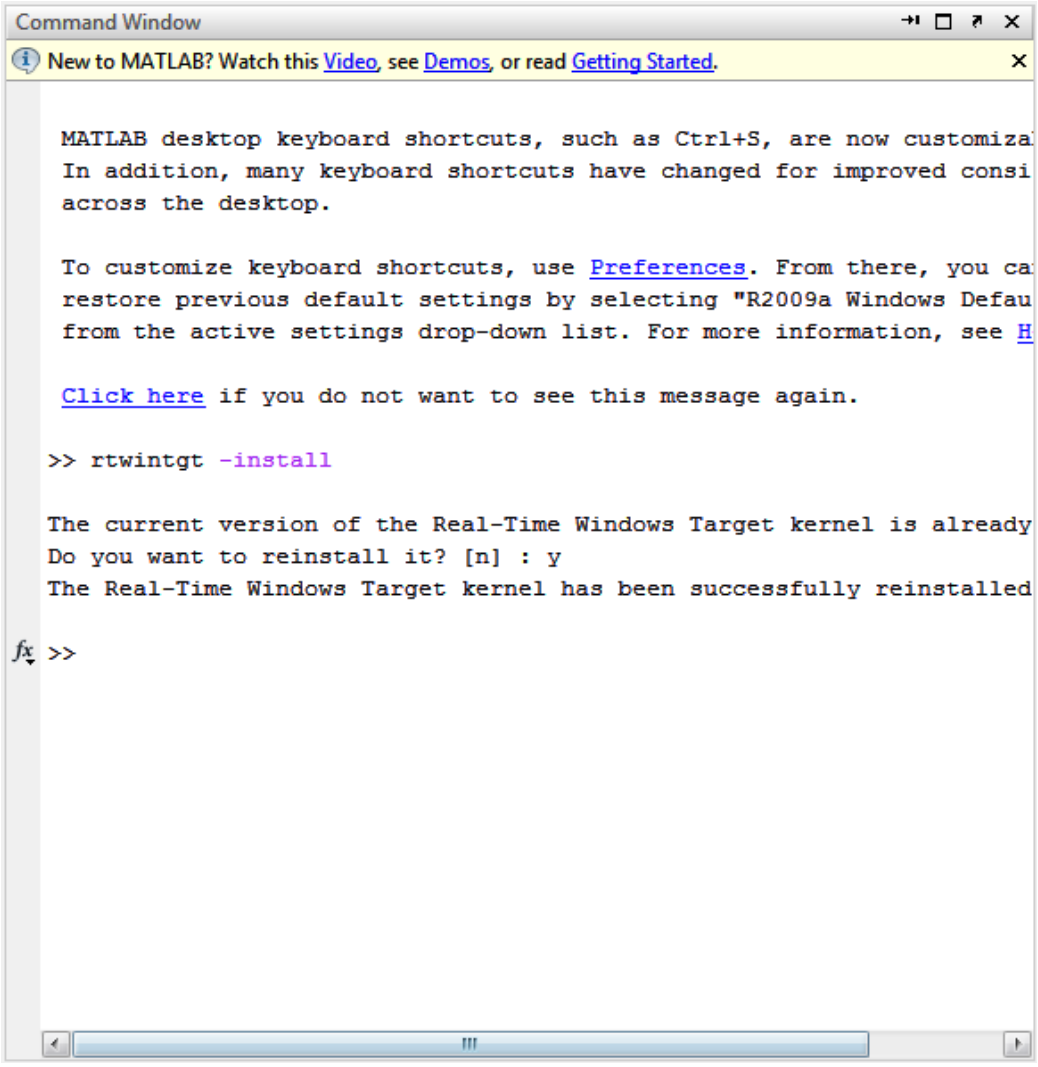

Fig 5.1: MATLAB Command Window

Step 2. Make new model. Add Packet Output block from Simulink Library

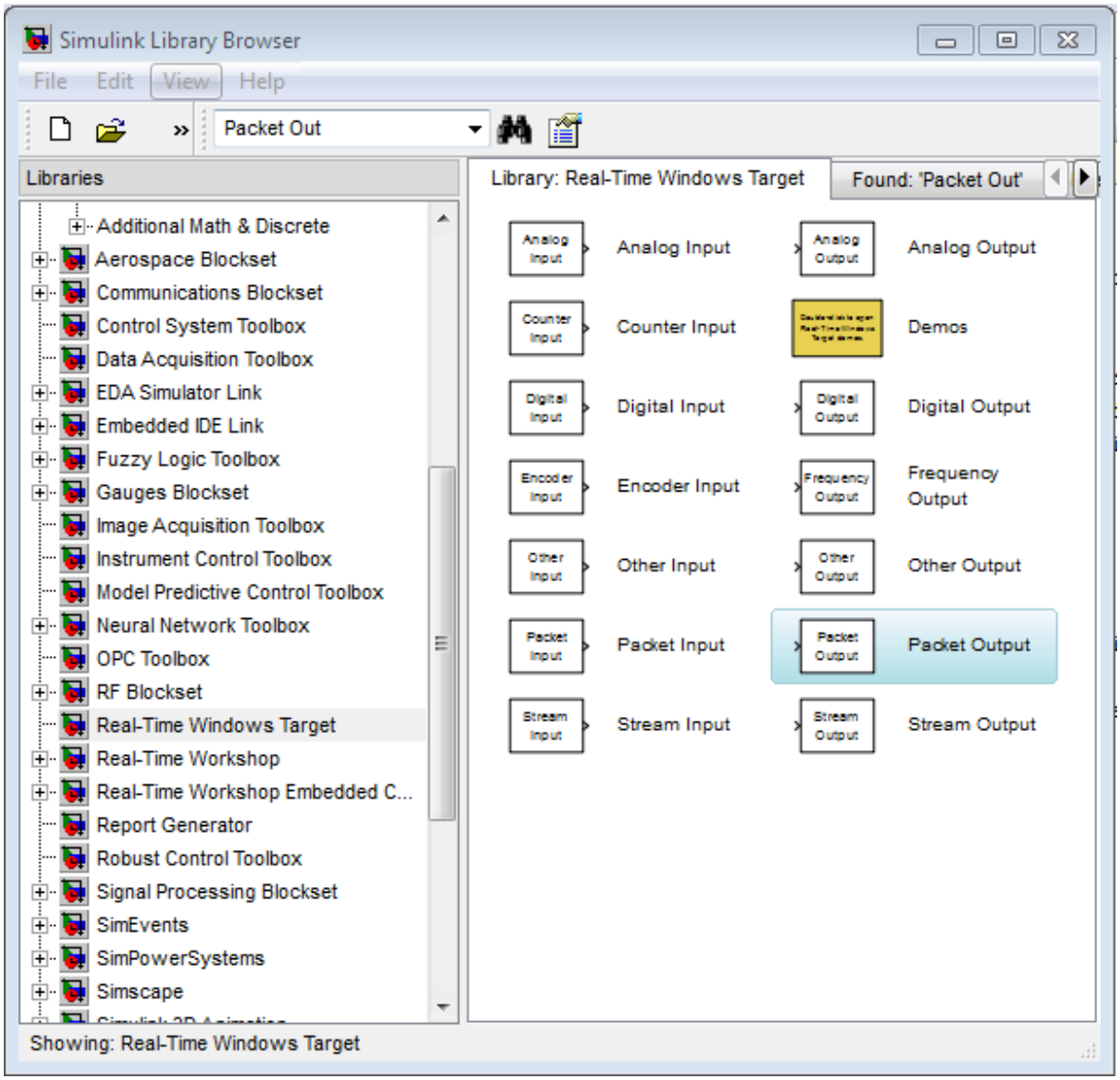

Fig 5.2: Simulink Library Browser

Step 3. Add Signal Generator Block from Simulink Library.

Step 4. Change Packet Output block parameters as follows –

I. Install new board > Standard Devices > UDP Protocol

II. Board Setup> enter local UDP port, ip address of the remote PC and the remote UDP port.

III. Sample Time as required.

IV. Output packet size – 8

V. Output packet field data types - '1*double'

The block parameters dialog box looks as shown below -

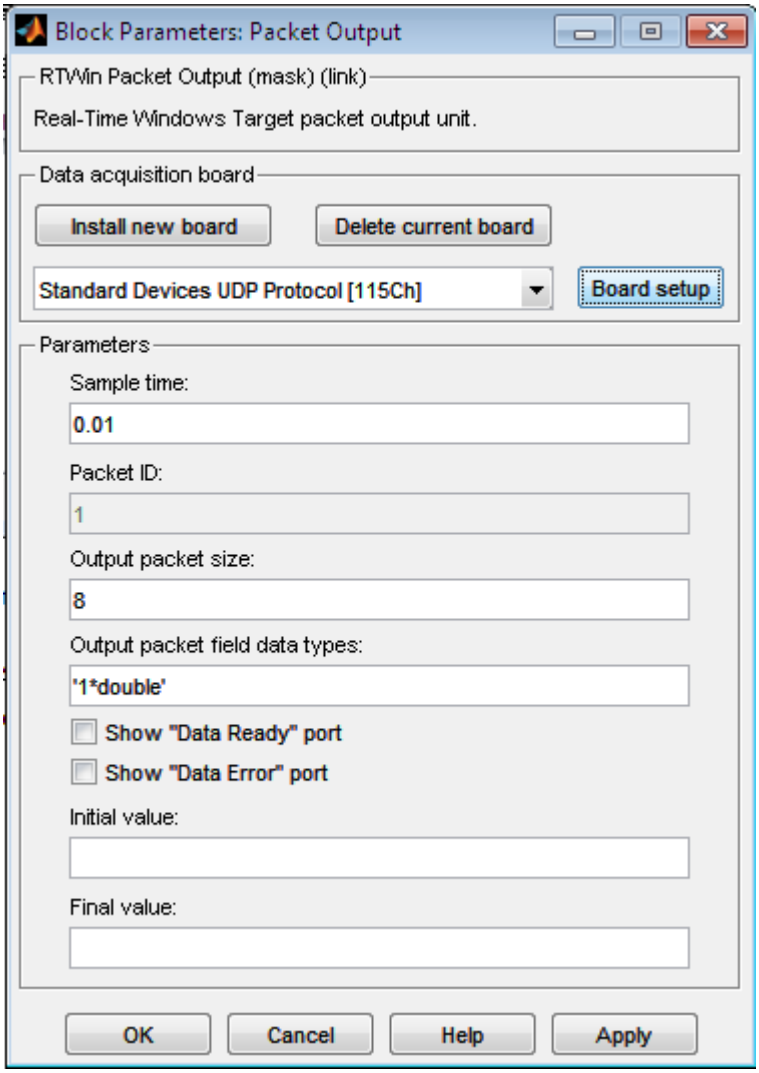

Fig 5.3: Packet Output Block Parameters

Board setup

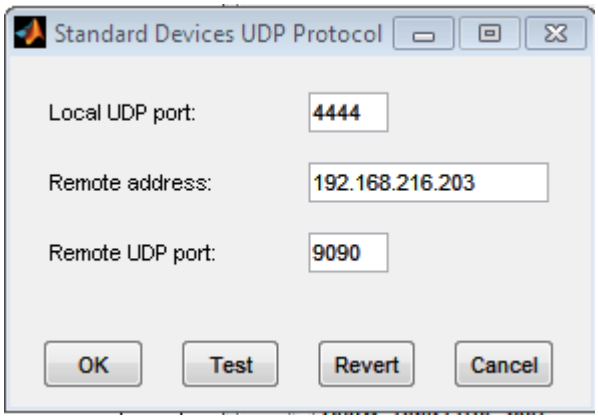

Fig 5.4: Board Setup Dialog Box

5.2 Steps for the Remote PC.

Follow Step 1 as mentioned in the previous case.

Step 2. Add a Packet Input block in the model and a scope.

Step 3. Configuration of UDP Packet Input block is similar to the output block.

The final model will look as shown below –

Local PC

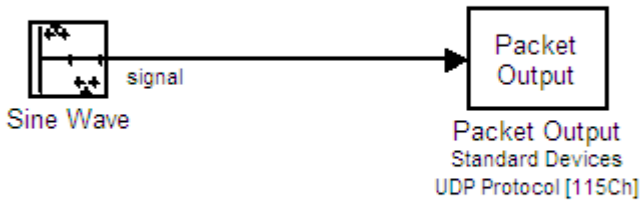

Fig 5.5: Local PC Simulink Model

Remote PC

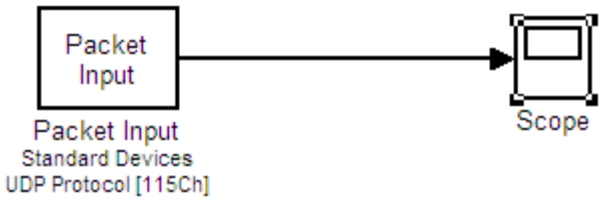

Fig 5.6: Remote PC Simulink Model

The simulation time is set to infinity, mode – External and configuration parameters are changed as –

I. Solver> Solver Options> Type> Fixed Step

II. Optimization> uncheck signal storage reuse checkbox.

III. Real Time Workshop> change the system target file to rtwin.tlc

Configuration parameters –

Solver

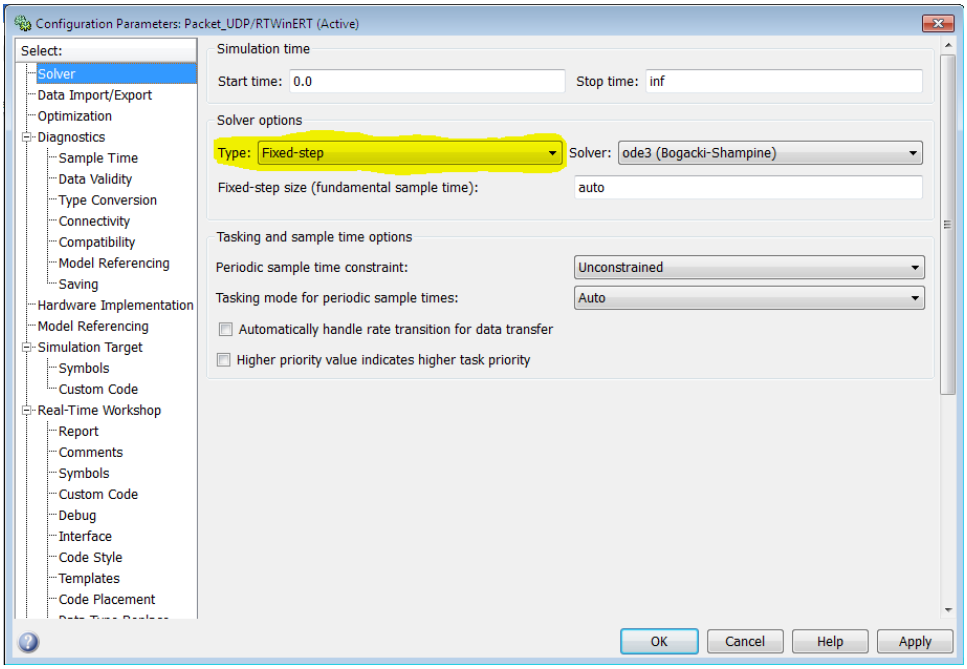

Fig 5.7: Configuration Parameters Setting - Solver

Optimization

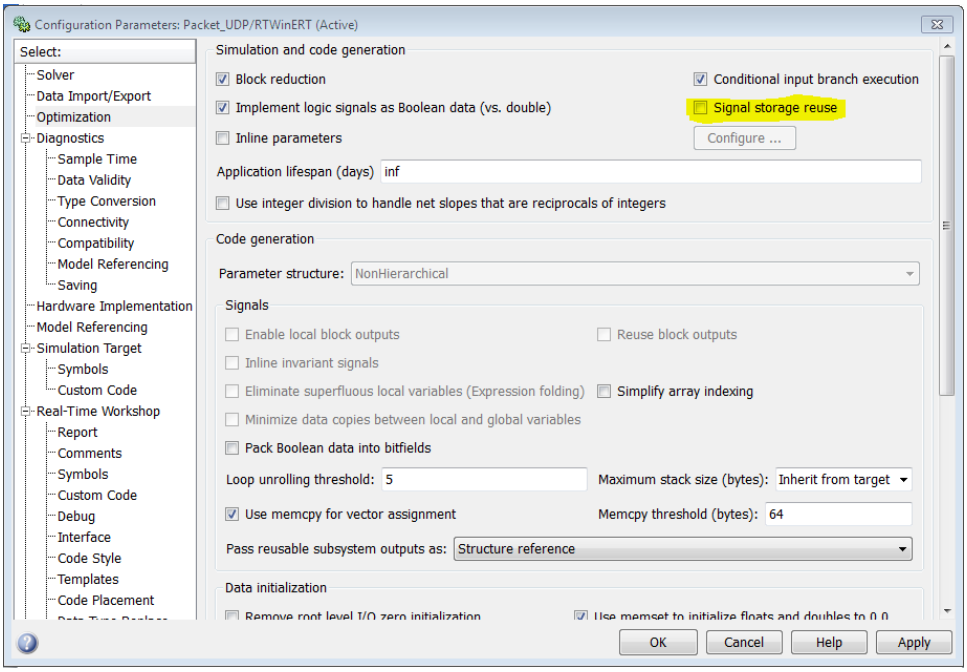

Fig 5.8: Configuration Parameters Setting - Optimization

Real Time Workshop

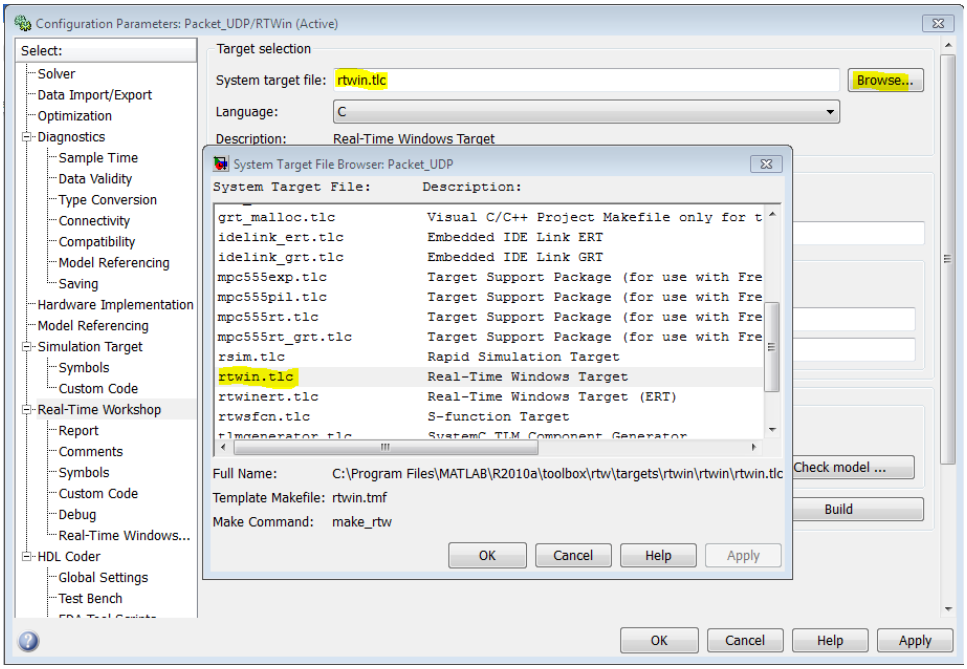

Fig 5.9: Configuration Parameters Setting – Real Time Workshop

This will enable us to set up communication between two PCs. When the models are run the remote PC is able to receive the signal sent with some delay.

A closed loop was formed and it was found that the data sent to and fro took **0.009 seconds** to complete the loop. The network characteristic is shown below -

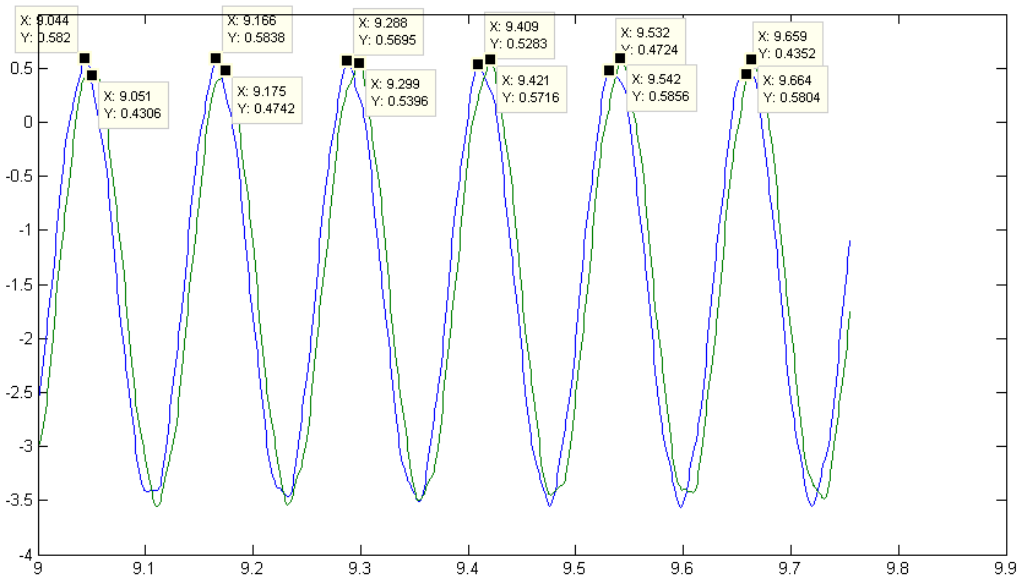

Fig 5.10: Network Delay Graph

Chapter 6. Remote control of DC servo motor using PID controller

6.1 Simulink model for local PC

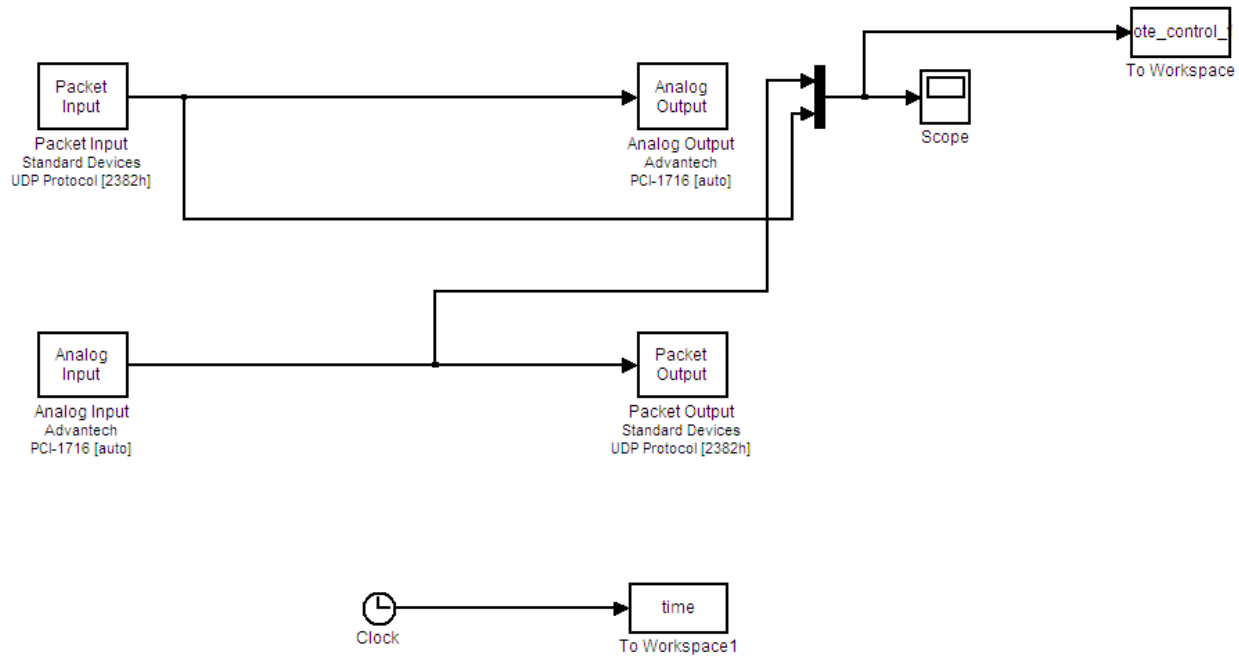

Fig 6.1: Simulink Model Local PC

Blocks used –

I. Packet IP/OP

II. Analog IP/OP

III. Scope (to compare the input signal with the speed output)

Here Packet IP/OP blocks are used to receive the control signal and send the speed output. Analog IP and Analog OP are the interfacing blocks for PCI card.

6.2 Simulink model for remote PC

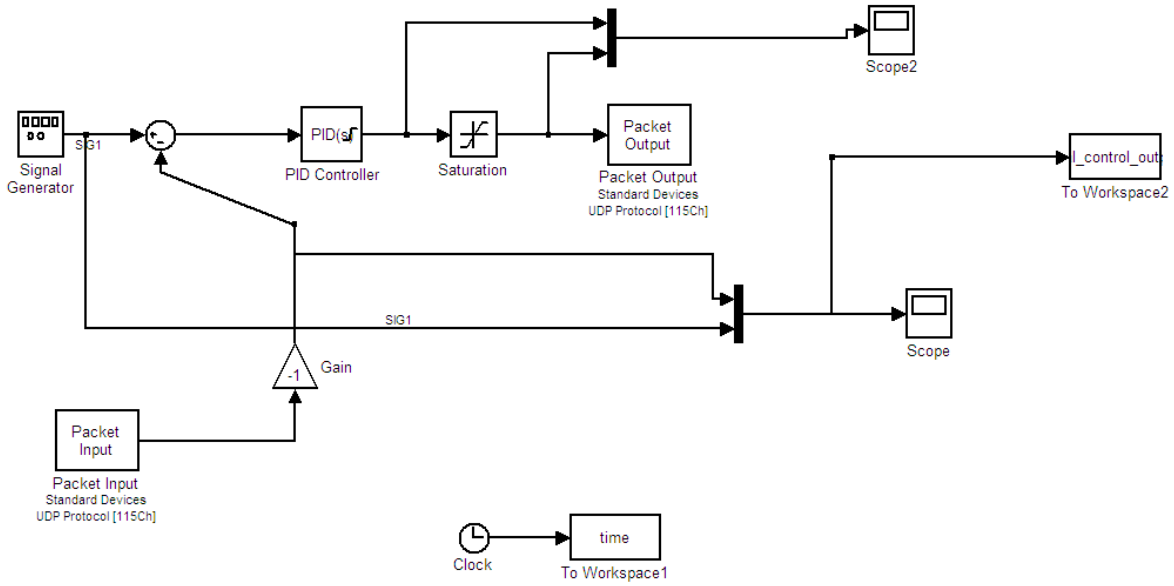

Fig 6.2: Simulink Model Remote PC

Blocks used –

I. Packet IP/OP

II. Analog IP/OP

III. PID

IV. Saturation (used to limit the PID gain)

V. Sum

VI. Gain

6.2.1 The remote control characteristics are shown below:

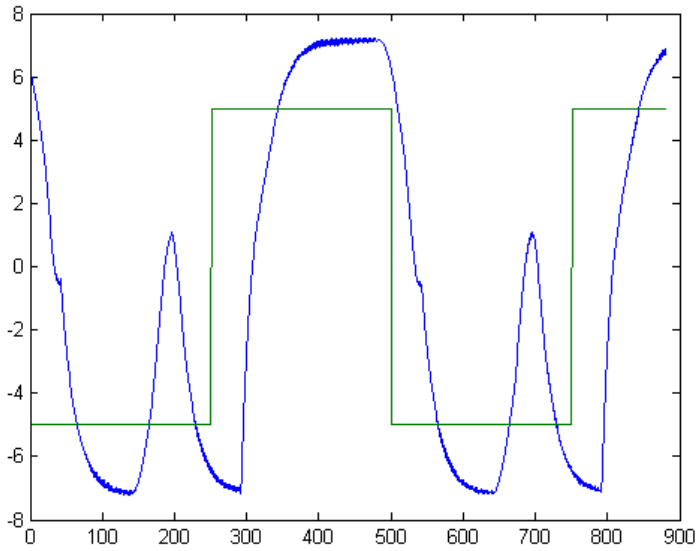

Fig 6.3: Control Characteristics (Frequency 0.2Hz)

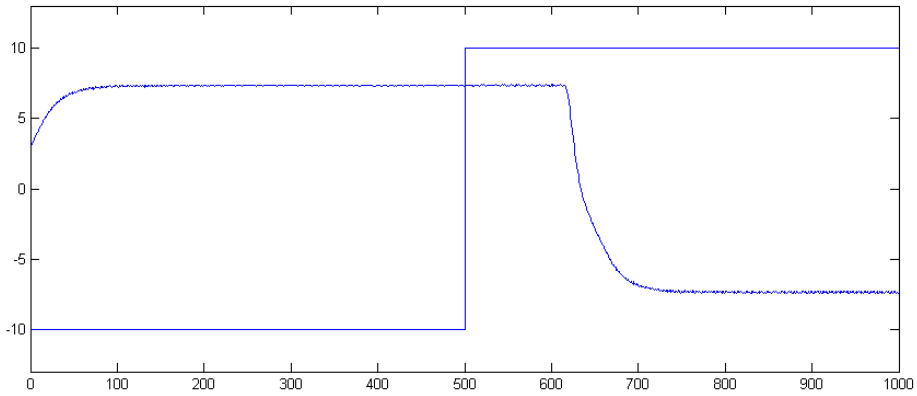

Fig 6.4: Control Characteristics (Frequency 0.2Hz)

A delay in the response can clearly be seen and proper speed control was not achieved.

To correct this, smith predictor was used.

Chapter 7. Remote control by use of smith predictor

7.1 Simulink model for remote PC

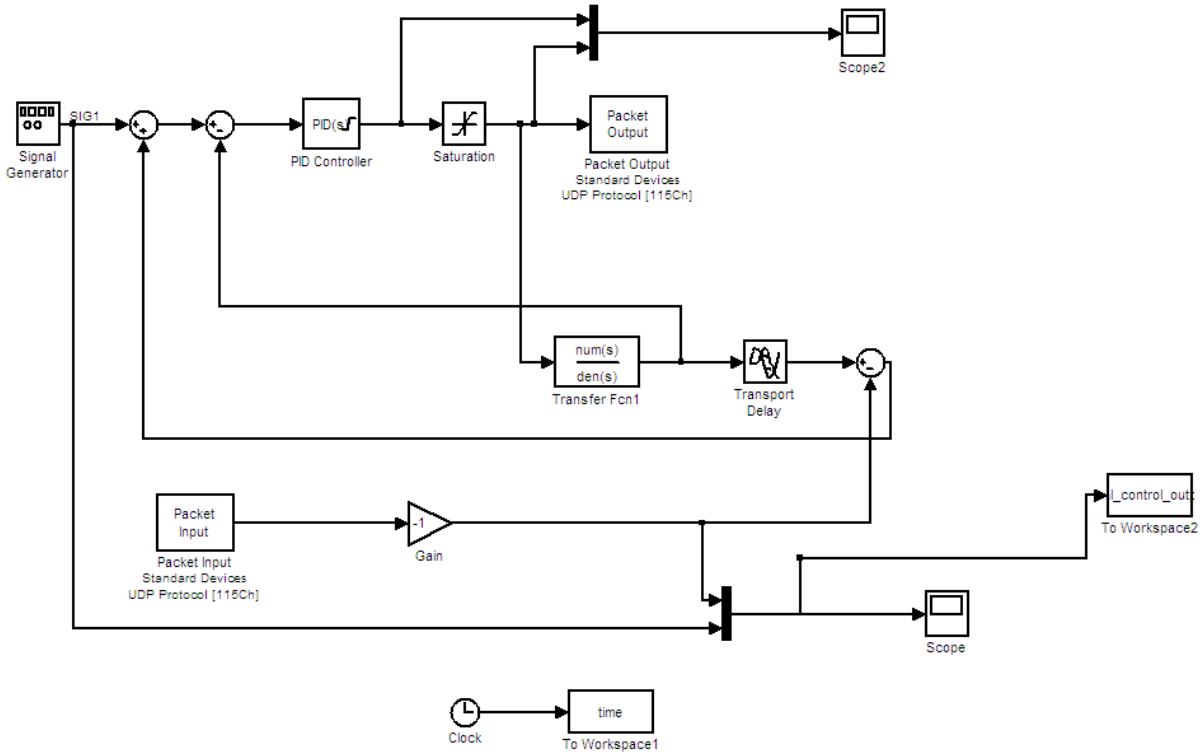

Fig 7.1: Smith Predictor Simulink Model

The transfer function used was derived earlier for design of PID controller. The transport delay could not be measured correctly and so the desired result was not obtained.

7.2 Simulink model for local PC

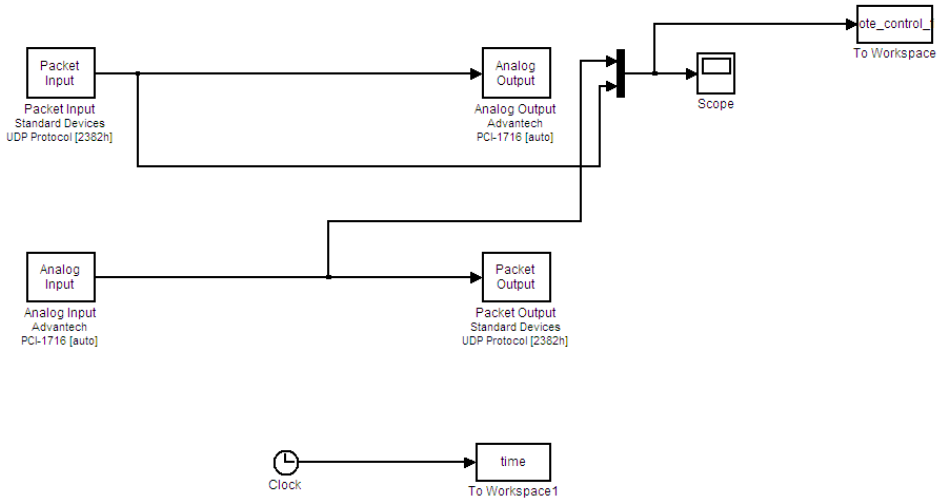

Fig 7.2: Local PC Simulink Model

Chapter 8. Conclusion And Scope For Future Work

8.1 Conclusion

This thesis presents study on MATLAB (2010 version) based real-time control implementation of the DC servo motor using PCI-1716.

It was observed that on running the motor in open loop, the speed output does not successfully track the reference. So, the need for a closed loop control system was realized.

To achieve this, the system was modeled using System Identification Toolbox of MATLAB, and a PID controller was designed for the model. Then the system was run in closed loop configuration. It was observed that the speed output of the closed loop control system successfully tracks the reference.

Then, to achieve remote control of the DC servo motor through PCI card, using MATLAB, a communication was set up between two computers based on UDP. The total time delay in sending a signal from one PC to another and receiving the same signal back was also calculated.

Finally, the motor speed was controlled from a remote computer using PID controller. It was observed that the communication network introduced delay in the system output with respect to the reference. These delays may have an effect on the stability and performance of the system.

8.2 Scope for future work

To compensate for the communication delay during remote control of the motor, Smith predictor may be used. The Simulink model for the same has been given in the thesis.

However, we have not been able to obtain satisfactory results using Smith Predictor. The necessary modifications may be made to the model (using Smith Predictor) for a more efficient remote control of the DC servo motor.

Chapter 9. Bibliography

MATLAB. (n.d.). Retrieved from http://www.kxcad.net/cae_MATLAB/toolbox/rtwin/ug/brb_e3z.html

MATLAB. (n.d.). *MATLAB Help*. Retrieved from http://www.mathworks.com/help/toolbox/rtwin/ref/packetoutput.html

Modeling of DC Servo Motor. (n.d.). Retrieved from http://www.eng.buffalo.edu/Courses/mae340/mae340Labs/Lab8a.htm

Servo. (n.d.). Retrieved from http://www.tpub.com/neets/book15/62.htm

Servo Motor Control. (n.d.). Retrieved from http://www.educypedia.be: http://kilowattclassroom.com/Archive/ClosedLoopControl.pdf

Wikipedia. (n.d.). *User Datagram Protocol*. Retrieved from http://en.wikipedia.org/: http://en.wikipedia.org/wiki/User_Datagram_Protocol