

**PID CONTROLLER DESIGN FOR CONTROLLING DC MOTOR  
SPEED USING MATLAB APPLICATION**

**MOHAMED FARID BIN MOHAMED FARUQ**

**UNIVERSITI MALAYSIA PAHANG**

UNIVERSITI MALAYSIA PAHANG

**BORANG PENGESAHAN STATUS TESIS♦**

**JUDUL:** **PID CONTROLLER DESIGN FOR CONTROLLING DC  
MOTOR SPEED USING MATLAB APPLICATION**

**SESI PENGAJIAN:** **2008/2009**

Saya **MOHAMED FARID BIN MOHAMED FARUQ ( 860724-56-5251 )**  
(HURUF BESAR)

mengaku membenarkan tesis (Sarjana Muda/~~Sarjana~~ /~~Doktor Falsafah~~)\* ini disimpan di Perpustakaan dengan syarat-syarat kegunaan seperti berikut:

1. Tesis adalah hakmilik Universiti Malaysia Pahang (UMP).
2. Perpustakaan dibenarkan membuat salinan untuk tujuan pengajian sahaja.
3. Perpustakaan dibenarkan membuat salinan tesis ini sebagai bahan pertukaran antara institusi pengajian tinggi.
4. \*\*Sila tandakan ( √ )

☐

**SULIT**

(Mengandungi maklumat yang berdarjah keselamatan atau kepentingan Malaysia seperti yang termaktub di dalam AKTA RAHSIA RASMI 1972)

☐

**TERHAD**

(Mengandungi maklumat TERHAD yang telah ditentukan oleh organisasi/badan di mana penyelidikan dijalankan)

☒

**TIDAK TERHAD**

Disahkan oleh:

\_\_\_\_\_  
(TANDATANGAN PENULIS)

\_\_\_\_\_  
(TANDATANGAN PENYELIA)

Alamat Tetap:

**29 JALAN RASMI JAYA,**  
**TAMAN RASMI JAYA,**  
**68000 AMPANG,**  
**SELANGOR**

**AHMAD NOR KASRUDDIN BIN NASIR**  
( Nama Penyelia )

Tarikh: **11 NOVEMBER 2008**

Tarikh: : **11 NOVEMBER 2008**

CATATAN:      \*      Potong yang tidak berkenaan.  
                 \*\*      Jika tesis ini SULIT atau TERHAD, sila lampirkan surat daripada pihak berkuasa/organisasi berkenaan dengan menyatakan sekali tempoh tesis ini perlu dikelaskan sebagai atau TERHAD.  
                 ♦      Tesis dimaksudkan sebagai tesis bagi Ijazah doktor Falsafah dan Sarjana secara Penyelidikan, atau disertasi bagi pengajian secara kerja kursus dan penyelidikan, atau Laporan Projek Sarjana Muda (PSM).

“I hereby acknowledge that the scope and quality of this thesis is qualified for the  
award of the Bachelor Degree of Electrical Engineering (Power System)”

Signature : \_\_\_\_\_

Name : AHMAD NOR KASRUDDIN BIN NASIR

Date : 11 NOVEMBER 2008

PID CONTROLLER DESIGN FOR CONTROLLING DC MOTOR  
SPEED USING MATLAB APPLICATION

MOHAMED FARID BIN MOHAMED FARUQ

This thesis is submitted as partial fulfillment of the requirements for the award of the  
Bachelor of Electrical Engineering (Power System)

Faculty of Electrical & Electronics Engineering  
Universiti Malaysia Pahang

NOVEMBER, 2008

“All the trademark and copyrights use herein are property of their respective owner. References of information from other sources are quoted accordingly; otherwise the information presented in this report is solely work of the author.”

Signature : \_\_\_\_\_

Author : MOHAMED FARID BIN MOHAMED FARUQ

Date : 11 NOVEMBER 2008

To my beloved mother, father and sister

## **ACKNOWLEDGEMENT**

In preparing this thesis, I was in contact with many people, researchers, academicians, and practitioners. They have contributed towards my understanding and thoughts. In particular, I wish to express my sincere appreciation to my thesis supervisor, Mr. Ahmad Nor Kasruddin Bin Nasir, for encouragement, guidance, critics and friendship. Without his continued support and interest, this thesis would not have been the same as presented here. I would like to give my sincere appreciation to all my friends and others who have provided assistance at various occasions. Their views and tips are useful indeed. Unfortunately, it is not possible to list all of them in this limited space. Finally to all my family members where without them I would not be here.

## **ABSTRACT**

This project is a simulation and experimental investigation into the development of PID controller using MATLAB/SIMULINK software. The simulation development of the PID controller with the mathematical model of DC motor is done using Ziegler–Nichols method and trial and error method. The PID parameter is to be tested with an actual motor also with the PID controller in MATLAB/SIMULINK software. In order to implement the PID controller from the software to the actual DC motor data acquisition is used. From the simulation and the experiment, the result performance of the PID controller is compared in term of response and the assessment is presented.



## ABSTRAK

Project in adalah penyelidikan secara simulasi dan eksperimen dalam pembangunan pengawal PID menggunakan perisian MATLAB/SIMULINK. Pembangunan simulasi pengawal PID dengan model matematik motor DC menggunakan kaedah Ziegler–Nichols dan kaedah cuba dan jaya. Parameter pengawal PID akan diuji dengan motor sebenar juga dengan pengawal PID menggunakan perisian MATLAB/SIMULIN. Bagi mengaplikasikan pengawal PID dari perisian kepada motor DC sebenar, *data acquisition card* di gunakan. Dari simulasi dan eksperimen, keputusan kecekapan dari pengawal PID dibandingkan dari segi respon dan analisis di lakukan dan dibentangkan.

## TABLE OF CONTENTS

CHAPTER	TITLE	PAGE
	<b>TITLE PAGE</b>	<b>i</b>
	<b>DECLARATION</b>	<b>ii</b>
	<b>DEDICATION</b>	<b>iii</b>
	<b>ACKNOWLEDGEMENT</b>	<b>iv</b>
	<b>ABSTRACT</b>	<b>v</b>
	<b>ABSTRAK</b>	<b>vi</b>
	<b>TABLE OF CONTENTS</b>	<b>vii</b>
	<b>LIST OF TABLES</b>	<b>x</b>
	<b>LIST OF FIGURES</b>	<b>xi</b>
	<b>LIST OF SYMBOLS</b>	<b>xv</b>
	<b>LIST OF APPENDICES</b>	<b>xvi</b>
 <b>I</b>	 <b>INTRODUCTION</b>	
1.1	Background of Project	1
1.2	Objective	2
1.3	Scope of Work	2
1.4	Problem Statement	3

## **II LITERATURE REVIEW**

2.1	Permanent Magnet Direct Current Motor	4
2.2	Control Theory	5
2.2.1	Closed-Loop Transfer Function	6
2.2.2	PID Controller	8
2.3	Pulse Width Modulation	9
2.4	MATLAB® and SIMULINK®	11

## **III METHODOLOGY**

3.1	System Description	15
3.1.1	Mathematical Model	19
3.2	Data Acquisition	22
3.2.1	PCI-1710HG	24
3.2.1.1	Specification	25
3.2.1.2	Installation Guide	29
3.3	Real Time Computing	31
3.4	Real Time Window Target	32
3.4.1	Setup and Configuration	34
3.4.1.1	Compiler	34
3.4.1.2	Kernel Setup	35
3.4.1.3	Testing the Installation	37
3.4.2	Creating a Real Time Application	39
3.4.3	Entering Configuration Parameters for Simulink	47
3.4.4	Entering Simulation Parameters for Real-Time Workshop	49
3.4.5	Creating a Real-Time Application	51
3.4.6	Running a Real-Time Application	52
3.5	Driver	53
3.5.1	Geckodrive G340	54

	3.5.2 Alternative Driver IR2109	55
	3.5 Project Planning	56
<b>IV</b>	<b>RESULT AND DISCUSSION</b>	
	4.1 Controller Design	57
	4.1.1 PID Controller	58
	4.1.1.1 Zeigler Nichols Method	58
	4.1.1.2 Trial and Error Method	59
	4.2 Simulation without PID Controller	60
	4.3 Simulation with PID Controller	61
	4.4 Experiment without PID Controller	62
	4.5 Experiments with PID Controller	66
<b>V</b>	<b>CONCLUSION AND RECOMENDATION</b>	
	5.1 Conclusion	70
	5.2 Future Recommendation	71
	<b>REFERENCES</b>	73
	<b>APPENDICES</b>	
	APPENDIX A	76
	APPENDIX B	77
	APPENDIX C	78

**LIST OF TABLES**

<b>TABLE NO.</b>	<b>TITLE</b>	<b>PAGE</b>
4.1	Typical Values of Proportional, Integral, and Derivative feedback Coefficient for PID-type Controller	59
4.2	Speed and Voltage for every 10% duty cycle	65

## LIST OF FIGURE

FIGURE NO.	TITLE	PAGE
2.1	Concept of the Feedback Loop to Control the Dynamic Behavior of the Reference	5
2.2	Closed-Loop Controller or Feedback Controller	7
2.3	A Square Wave, Showing the Definitions of $y_{min}$ , $y_{max}$ and D	9
2.4	PWM Pulse Generate from Comparing Sinewave and Sawtooth	10
2.7	MATLAB <sup>®</sup> Default Command Windows	12
2.8	SIMULINK <sup>®</sup> Running a Simulation of a Thermostat-Controlled Heating System	14
3.1	Block Diagram of the System	15
3.2	Geckodrive G340	16
3.3	Alternative Driver (IR2109)	16

3.4	Power Supply	16
3.5	Oscilloscope	16
3.6	Data Acquisition Card (PCI-1710HG)	16
3.7	Industrial Wiring Terminal Board with CJC Circuit (PCLD-8710)	17
3.8	Personal Computer	17
3.9	Litton - Clifton Precision Servo DC Motor JDH-2250	18
3.10	Schematic Diagram of the DC Motor	19
3.11	Block Diagram of the Open-Loop Permanent-Magnet DC Motor	21
3.12	Block Diagram of the Open-Loop Servo Actuated by Permanent-Magnet DC Motor	21
3.13	Block Diagram of the Closed-Loop Servo with PID Controller	22
3.14	Pin Assignment	27
3.15	Block Diagram of PCI-1710HG	28
3.16	PCI-1710HG Installation Flow Chart	30
3.17	Simulink Model rtvdp.mdl	37

3.18	Output Signal rtvdp.mdl	39
3.19	Simulink Library Browser	40
3.20	Empty Simulink Windows	40
3.21	Signal Generator Block Parameter	41
3.22	Analog Output Block Parameter	42
3.23	Board Test OK Dialog	43
3.24	Scope Parameters Dialog Box	45
3.25	Scope Window	46
3.26	Scope Properties: axis 1	46
3.27	Completed Simulink Block Diagram	47
3.28	Configuration Parameter (Solver) Windows.	48
3.29	Configuration Parameter (Hardware Implementation) Windows	49
3.30	System Target File Browsers.	50
3.31	Configuration Parameter (Real-Time Workshop) Windows	50
3.32	Connect To Target and Start Real-Time Code	52
3.33	Geckodrive G340 Block Diagram	54



3.34	Typical Connections for IR2109	55
3.35	Flow Chart of Project	56
4.1	Simulink Block of PID Controller	58
4.2	Detailed Simulink Block of the System	60
4.3	Output of DC Motor without PID Controller	60
4.4	Detail Simulink Block of the System with PID Controller	61
4.5	Output of DC Motor without PID Controller	62
4.6	Simulink Block of Experiment without PID	63
4.7	10% Duty Cycle Pulse	63
4.8	50% Duty Cycle Pulse	64
4.9	90% Duty Cycle Pulse	64
4.10	Velocity Estimation	65
4.11	Complete Simulink Block of the Experiment	67
4.12	Velocity Decoder Subsystem Simulink Block	68

## LIST OF SYMBOLS

$D$	-	duty cycle
$T$	-	period
$T_L$	-	load torque
$\Theta_r$	-	angle
$\omega_r$	-	rotor angular displacement
$i_a$	-	armature current
$E_a$	-	Induced emf
$k_a$	-	back emf / torque constant
$r_a$	-	armature resistance
$L_a$	-	armature inductance
$J$	-	moment of inertia
$B_m$	-	viscous friction coefficient
$T_{\text{viscous}}$	-	viscous friction torque
$u_a$	-	armature voltage
$k_p$	-	proportional coefficient
$k_i$	-	integral coefficient
$k_d$	-	derivative coefficient
$T_{\text{ocs}}$	-	period of self-sustained oscillation
$k_{p\text{max}}$	-	critical value of proportional coefficient

**LIST OF APPENDICES**

<b>APPENDIX</b>	<b>TITLE</b>	<b>PAGE</b>
A	Simulink Block of PID Control DC Motor (Simulation) Simulink Block of DC Motor Simulink Block of PID Controller	76
B	Simulink Block of PID Control DC Motor (Experiment) Simulink Block of Velocity Decoder	77
C	Embedded MATLAB Function MATLAB Command	78

## **CHAPTER 1**

### **INTRODUCTION**

#### **1.1 Background of Project**

Permanent magnet direct current motor (PMDC) have been widely use in high-performance electrical drives and servo system. There are many difference DC motor types in the market and all with it good and bad attributes. Such bad attribute is the lag of efficiency. In order to overcome this problem a controller is introduce to the system.

There are also many types of controller used in the industry, such controller is PID controller. PID controller or proportional–integral–derivative controller is a generic control loop feedback mechanism widely used in industrial control systems. A PID controller attempts to correct the error between a measured process variable and a desired set point by calculating and then outputting a corrective action that can adjust the process accordingly. So by integrating the PID controller to the DC motor were able to correct the error made by the DC motor and control the speed or the position of the motor to the desired point or speed.

## **1.2 Objective**

The objectives of this project are:

- i. To fulfill the requirement for the subject BEE4712: Engineering Project.
- ii. To explore and apply the knowledge gain in lectures into practical applications.
- iii. To control the speed of DC motor with PID controller using MATLAB/SIMULINK application.
- iv. To design the PID controller and tune it using MATLAB/SIMULINK.
- v. To compare and analyze the result between the simulation result using a DC motor mathematical model in MATLAB/SIMULINK and the experimental result using the actual motor.

## **1.3 Scope of Work**

The scope of this project is;

- i. Design and produce the simulation of the PID controller
- ii. Simulate the PID controller with the modeling of the DC motor
- iii. Implement the PID simulation with an actual DC motor
- iv. The comparison of the simulation result with the actual DC motor

#### **1.4 Problem Statement**

The problem encounter when dealing with DC motor is the lag of efficiency and losses. In order to eliminate this problem, controller is introduce to the system. There's few type of controller but in this project, PID controller is chosen as the controller for the DC motor. This is because PID controller helps get the output, where we want it in a short time, with minimal overshoot and little error.

## **CHAPTER 2**

### **LITERATURE REVIEW**

#### **2.1 Permanent Magnet Direct Current Motor**

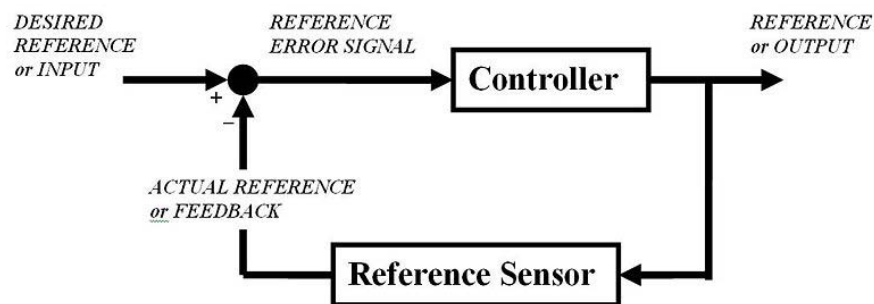
A DC motor is designed to run on DC electric power [3]. An example is Michael Faraday's homopolar motor, and the ball bearing motor. There are two types of DC motor which are brush and brushless types, in order to create an oscillating AC current from the DC source and internal and external commutation is use respectively. So they are not purely DC machines in a strict sense [3].

A brushless DC motor (BLDC) is a synchronous electric motor which is powered by direct-current electricity (DC) and which has an electronically controlled commutation system, instead of a mechanical commutation system based on brushes [4]. In such motors, current and torque, voltage and rpm are linearly related [4]. BLDC has its own advantages such as higher efficiency and reliability, reduced noise, longer lifetime, elimination of ionizing sparks from the commutator, and overall reduction of electromagnetic interference (EMI). With no windings on the rotor, they are not subjected to centrifugal forces, and because the electromagnets are located around the perimeter, the electromagnets can be cooled by conduction to the motor casing, requiring no airflow inside the motor for cooling [4]. The disadvantage

is higher cost, because of two issues. First, it requires complex electronic speed controller to run.

## 2.2 Control Theory

Control theory is an interdisciplinary branch of engineering and mathematics that deals with the behavior of dynamical systems [7]. The desired output of a system is called the *reference* [7]. When one or more output variables of a system need to follow a certain reference over time, a controller manipulates the inputs to a system to obtain the desired effect on the output of the system [7].



**Figure 2.1** Concept of the Feedback Loop to Control the Dynamic Behavior of the Reference

If we consider an automobile cruise control, it is design to maintain the speed of the vehicle at a constant speed set by the driver. In this case the system is the vehicle. The vehicle speed is the output and the control is the vehicle throttle which influences the engine torque output. One way to implement cruise control is by locking the throttle at the desired speed but when encounter a hill the vehicle will slow down going up and accelerate going down. In fact, any parameter different than what was assumed at design time will translate into a proportional error in the output velocity, including exact mass of the

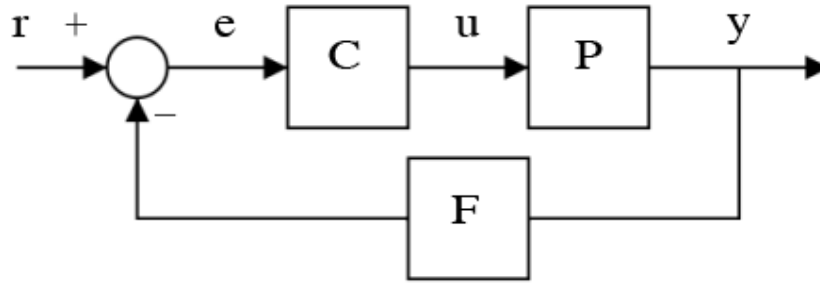


vehicle, wind resistance, and tire pressure [7]. This type of controller is called an open-loop controller because there is no direct connection between the output of the system (the engine torque) and the actual conditions encountered; that is to say, the system does not and cannot compensate for unexpected forces [7].

For a closed-loop control system, a sensor will monitor the vehicle speed and feedback the data to its computer and continuously adjusting its control input or the throttle as needed to ensure the control error to a minimum therefore maintaining the desired speed of the vehicle. Feedback on how the system is actually performing allows the controller (vehicle's on board computer) to dynamically compensate for disturbances to the system, such as changes in slope of the ground or wind speed [7]. An ideal feedback control system cancels out all errors, effectively mitigating the effects of any forces that may or may not arise during operation and producing a response in the system that perfectly matches the user's wishes [7].

### 2.2.1 Closed-Loop Transfer Function

The output of the system  $y(t)$  is fed back through a sensor measurement  $F$  to the reference value  $r(t)$ . The controller  $C$  then takes the error  $e$  (difference) between the reference and the output to change the inputs  $u$  to the system under control  $P$ . This is shown in the figure. This kind of controller is a closed-loop controller or feedback controller. This is called a single-input-single-output (*SISO*) control system; *MIMO* (i.e. Multi-Input-Multi-Output) systems, with more than one input/output, are common. In such cases variables are represented through vectors instead of simple scalar values. For some distributed parameter systems the vectors may be infinite-dimensional (typically functions).



**Figure 2.2** Closed-loop controller or feedback controller

If we assume the controller  $C$ , the plant  $P$ , and the sensor  $F$  are linear and time-invariant (i.e.: elements of their transfer function  $C(s)$ ,  $P(s)$ , and  $F(s)$  do not depend on time), the systems above can be analyzed using the Laplace transform on the variables. This gives the following relations:

$$Y(s) = P(s)U(s)$$

$$U(s) = C(s)E(s)$$

$$E(s) = R(s) - F(s)Y(s).$$

Solving for  $Y(s)$  in terms of  $R(s)$  gives:

$$Y(s) = \left( \frac{P(s)C(s)}{1 + F(s)P(s)C(s)} \right) R(s) = H(s)R(s).$$

$$H(s) = \frac{P(s)C(s)}{1 + F(s)P(s)C(s)}$$

The expression is referred to as the closed-loop transfer function of the system. The numerator is the forward (open-loop) gain from  $r$  to  $y$ , and the denominator is one plus the gain in going around the feedback loop, the so-called loop gain. If  $|P(s)C(s)| \gg 1$ , i.e. it has a large norm with each value of  $s$ , and if  $|F(s)| \approx 1$ , then  $Y(s)$  is approximately equal to  $R(s)$ . This means simply setting the reference controls the output.

### 2.2.2 PID Controller

PID Control (proportional-integral-derivative) is by far the widest type of automatic control used in industry. Even though it has a relatively simple algorithm/structure, there are many subtle variations in how it is applied in industry [5]. A proportional–integral–derivative controller (PID controller) is a generic control loop feedback mechanism widely used in industrial control systems [1]. A PID controller will correct the error between the output and the desired input or set point by calculating and give an output of correction that will adjust the process accordingly. A PID controller has the general form

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

Where  $K_p$  is proportional gain,  $K_i$  is the integral gain, and  $K_d$  is the derivative gain.

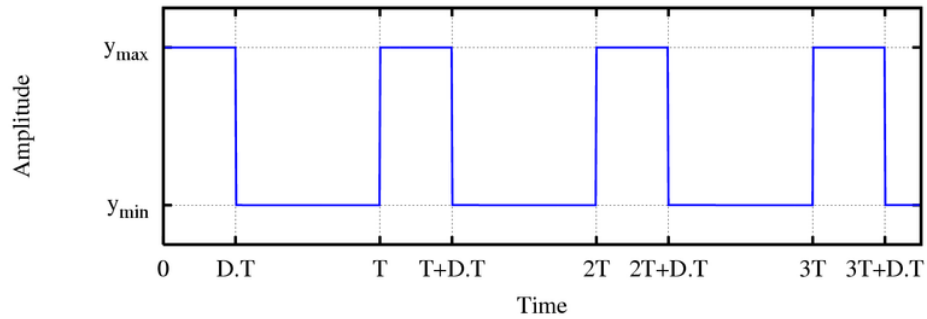
The PID controller calculation (algorithm) involves three separate parameters; the Proportional, the Integral and Derivative values [1]. The Proportional value determines the reaction to the current error, the Integral determines the reaction based on the sum of recent errors and the Derivative determines the reaction to the rate at which the error has been changing [1]. The weighted sum of these three actions is used to adjust the process via a control element such as the position of a control valve, the power supply of a heating element or DC motor speed and position.

### 2.3 Pulse Width Modulation

Pulse-width modulation (PWM) of a signal or power source involves the modulation of its duty cycle, to either convey information over a communications channel or control the amount of power sent to a load.

Pulse-width modulation uses a square wave whose pulse width is modulated resulting in the variation of the average value of the waveform. If we consider a square waveform  $f(t)$  with a low value  $y_{min}$ , a high value  $y_{max}$  and a duty cycle  $D$  (see figure 2.3), the average value of the waveform is given by:

$$\bar{y} = \frac{1}{T} \int_0^T f(t) dt.$$



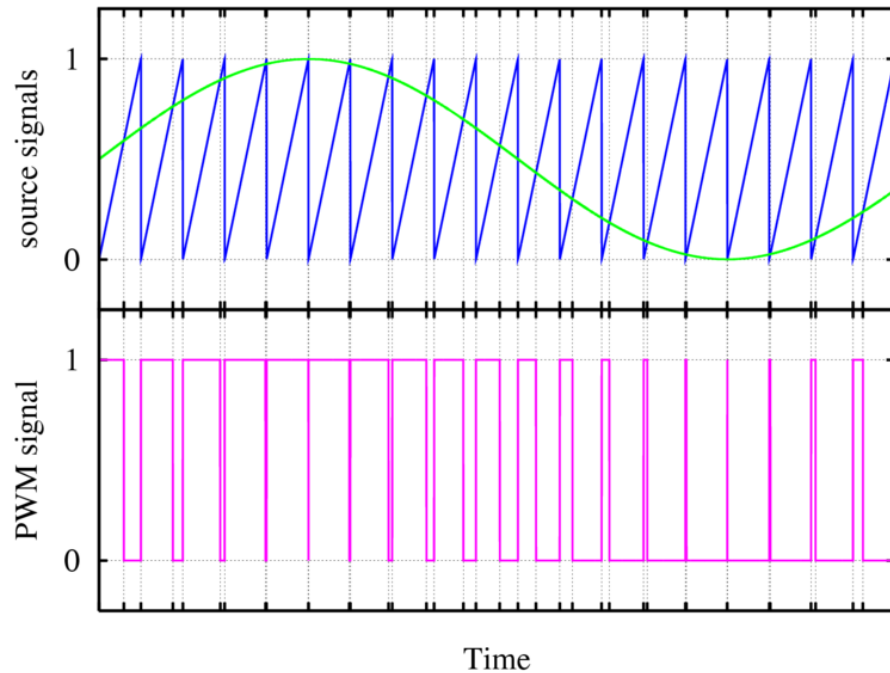
**Figure 2.3** A Square Wave, Showing the Definitions of  $y_{min}$ ,  $y_{max}$  and  $D$

As  $f(t)$  is a square wave, its value is  $y_{max}$  for  $0 < t < D \cdot T$  and  $y_{min}$  for  $D \cdot T < t < T$ . The above expression then becomes:

$$\begin{aligned} \bar{y} &= \frac{1}{T} \left( \int_0^{DT} y_{max} dt + \int_{DT}^T y_{min} dt \right) \\ &= \frac{D \cdot T \cdot y_{max} + T(1-D)y_{min}}{T} \\ &= D \cdot y_{max} + (1 - D) y_{min}. \end{aligned}$$

This latter expression can be fairly simplified in many cases where  $y_{min} = 0$  as  $\bar{y} = D \cdot y_{max}$ . From this, it is obvious that the average value of the signal ( $\bar{y}$ ) is directly dependent on the duty cycle  $D$ .

The simplest way to generate a PWM signal is the intersective method, which requires only a sawtooth or a triangle waveform (easily generated using a simple oscillator) and a comparator. When the value of the reference signal (the green sine wave in figure 2.4) is more than the modulation waveform (blue), the PWM signal (magenta) is in the high state, otherwise it is in the low state.



**Figure 2.4** PWM Pulse Generate from Comparing Sinewave and Sawtooth

## 2.4 MATLAB and SIMULINK

MATLAB is a high-performance language for technical computing. It integrates computation, visualization, and programming in an easy-to-use environment where problems and solutions are expressed in familiar mathematical notation. Typical uses include:

- Math and computation
- Algorithm development
- Data acquisition
- Modeling, simulation, and prototyping
- Data analysis, exploration, and visualization
- Scientific and engineering graphics
- Application development, including graphical user interface building

MATLAB is an interactive system whose basic data element is an array that does not require dimensioning. This allows you to solve many technical computing problems, especially those with matrix and vector formulations, in a fraction of the time it would take to write a program in a scalar non-interactive language such as C or Fortran.

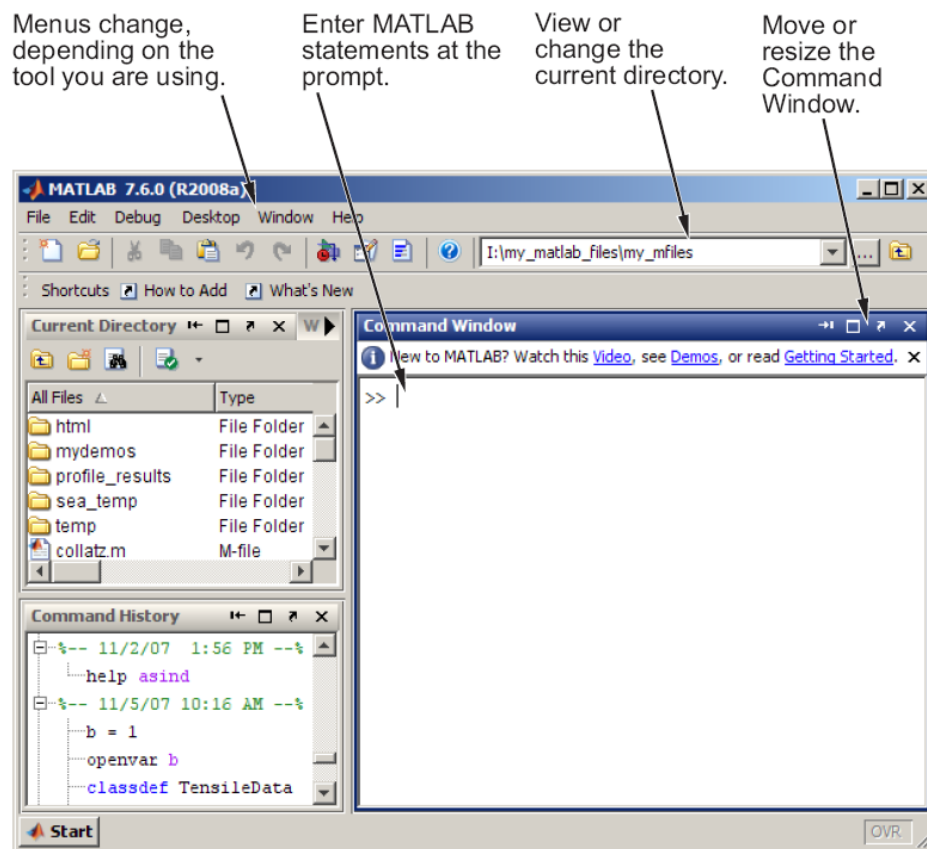
The name MATLAB stands for matrix laboratory. MATLAB was originally written to provide easy access to matrix software developed by the LINPACK and EISPACK projects. Today, MATLAB engines incorporate the LAPACK and BLAS libraries, embedding the state of the art in software for matrix computation.

MATLAB has evolved over a period of years with input from many users. In university environments, it is the standard instructional tool for introductory and advanced courses in mathematics, engineering, and science. In industry, MATLAB is the tool of choice for high-productivity research, development, and analysis.

MATLAB features a family of add-on application-specific solutions called toolboxes. Very important to most users of MATLAB, toolboxes allow you to learn

and apply specialized technology. Toolboxes are comprehensive collections of MATLAB functions (M-files) that extend the MATLAB environment to solve particular classes of problems. Areas in which toolboxes are available include signal processing, control systems, neural networks, fuzzy logic, wavelets, simulation, and many others.

When you start MATLAB, the MATLAB desktop appears, containing tools (graphical user interfaces) for managing files, variables, and applications associated with MATLAB. The following illustration shows the default desktop. You can customize the arrangement of tools and documents to suit your needs.



**Figure 2.7** MATLAB Default Command Windows

Simulink is software for modeling, simulating, and analyzing dynamic systems. Simulink enables you to pose a question about a system, model it, and see what happens.

With Simulink, you can easily build models from scratch, or modify existing models to meet your needs. Simulink supports linear and nonlinear systems, modeled in continuous time, sampled time, or a hybrid of the two. Systems can also be multirate — having different parts that are sampled or updated at different rates.

Thousands of scientists and engineers around the world use Simulink® to model and solve real problems in a variety of industries, including:

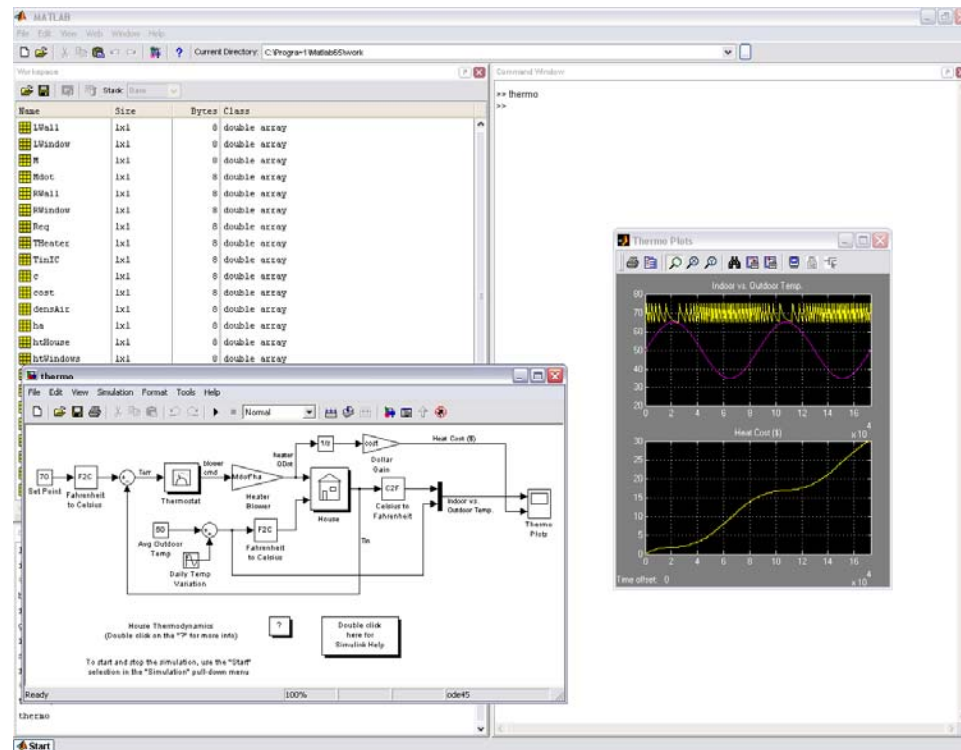
- Aerospace and Defense
- Automotive
- Communications
- Electronics and Signal Processing
- Medical Instrumentation

Model analysis tools include linearization and trimming tools, which can be accessed from the MATLAB command line, plus the many tools in MATLAB and its application toolboxes. Because MATLAB® and Simulink are integrated; you can simulate, analyze, and revise your models in either environment at any point.

Simulink® is tightly integrated with MATLAB. It requires MATLAB to run, depending on MATLAB to define and evaluate model and block parameters. Simulink® can also utilize many MATLAB features. For example, Simulink can use MATLAB to:

- Define model inputs.
- Store model outputs for analysis and visualization.
- Perform functions within a model, through integrated calls to MATLAB operators and functions.



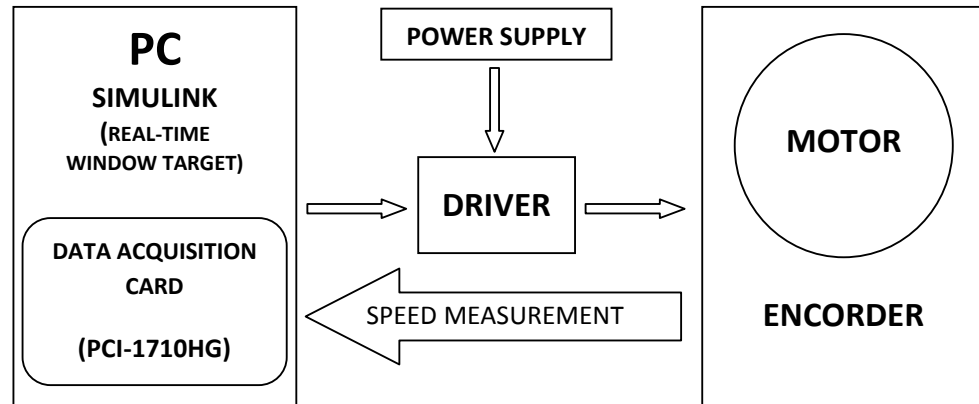


**Figure 2.8** Simulink Running a Simulation of a Thermostat-Controlled Heating System

## CHAPTER 3

### METHODOLOGY

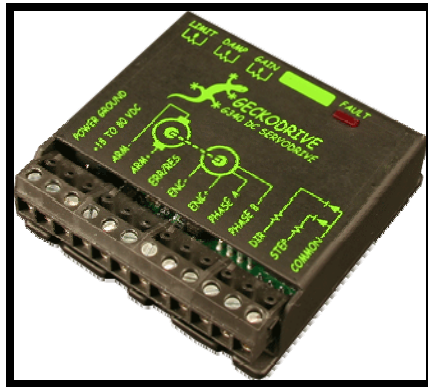
#### 3.1 System Description



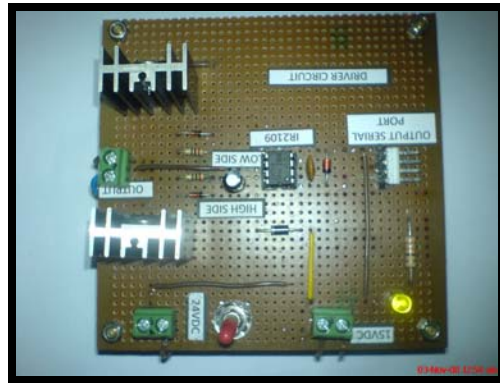
**Figure 3.1** Block Diagram of the System

The system block diagram is as shown in Figure 3.1. It consists of 2 main blocks (PC and Motor) that are connected through a driver and supplied by a power supply. The control algorithm is built in the Matlab/Simulink software and compiled with Real-Time Window Target. The Real-Time Window Target Toolbox includes an analog input and analog output that provide connection between the data acquisition card (PCI-1710HG) and the simulink model. For example, the speed of the DC motor could be controlled by supplying certain voltage and frequency from

signal generator block to the analog output in Simulink. From the analog input, the square received is displayed in a scope. The square wave pulse then is derived using the velocity equation to get the velocity of the DC motor speed. The speed acquired and the signal send can create a closed loop system with PID controller to control the speed of the DC motor. Figure 3.2 to Figure 3.9 shows the DC motor, driver, and other hardware used in this project and the DC motor specification.



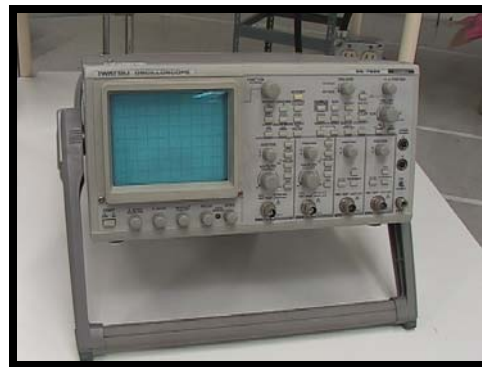
**Figure 3.2** Geckodrive G340



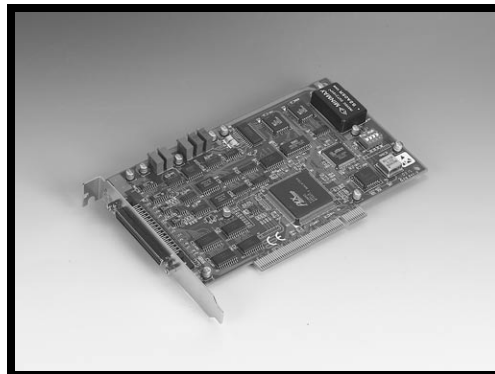
**Figure 3.3** Alternative Driver (IR2109)



**Figure 3.4** Power Supply



**Figure 3.5** Oscilloscope



**Figure 3.6** Data Acquisition Card (PCI-1710HG)



**Figure 3.7** Industrial Wiring Terminal Board with CJC Circuit (PCLD-8710)



**Figure 3.8** Personal Computer

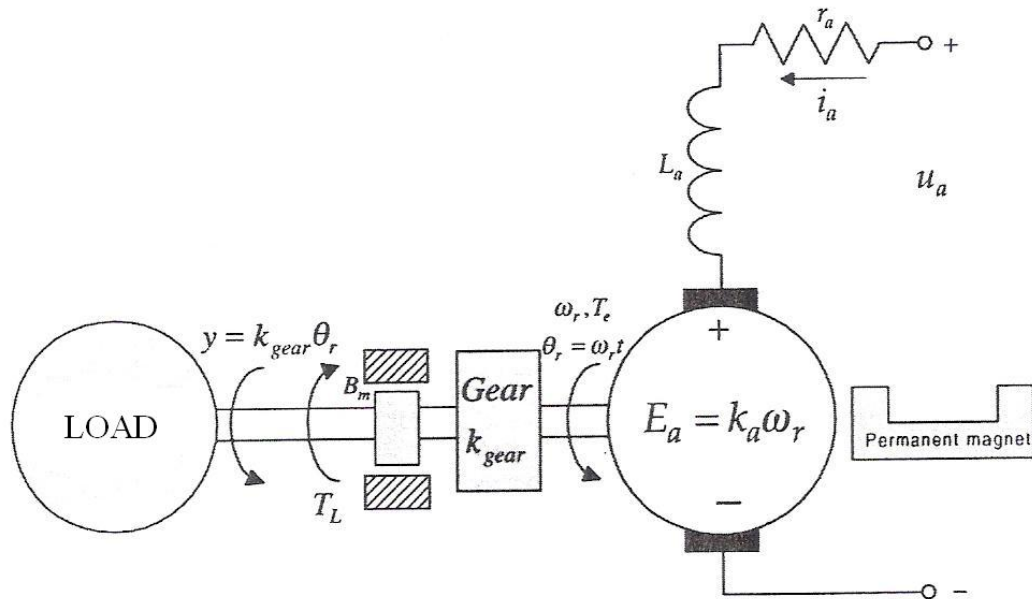


**Figure 3.9** Litton - Clifton Precision Servo DC Motor JDH-2250

**Specification of JDH-2250**

Torque Constant:	15.76 oz-in. / A
Back EMF:	11.65 VDC / KRPM
Peak Torque:	125 oz-in.
Cont. Torque:	16.5 oz-in.
Encoder:	250 counts / rev.
Channels:	A, B in quadrature, 5 VDC input (no index)
Body Dimensions:	2.25" dia. x 4.35" L (includes encoder)
Shaft Dimensions:	8 mm x 1.0" L w/flat

### 3.1.1 Mathematical Model



**Figure 3.10** Schematic Diagram of the DC Motor

To find the transfer function for the block diagram of the open and closed loop system a differential equation to describe the system dynamic. Kirchhoff's voltage is use to map the armature circuitry dynamic of the motor.

$$\frac{di_a}{dt} = \frac{r_a}{L_a} i_a - \frac{k_a}{L_a} \omega_r - \frac{1}{L_a} u_a$$

Using Newton's 2<sup>nd</sup> law

$$\sum \vec{T} = J \ddot{\theta} = J \frac{d\omega}{dt}$$

The electromagnetic torque developed by the permanent-magnet DC motor

$$T_e = k_a i_a$$

The viscous friction torque

$$T_{\text{viscous}} = B_m \omega_r$$

The load torque is denoted as  $T_L$ . Use the Newton's second law, we have

$$\frac{d\omega_r}{dt} = \frac{1}{J} (T_e - T_{\text{viscous}} - T_L) = \frac{1}{J} k_a i_a - B_m \omega_r - T_L$$

The dynamics of the rotor angular displacement

$$\frac{d\theta_r}{dt} = \omega_r$$

To find the transfer function, the derived three first-order differential equation

$$\frac{di_a}{dt} = \frac{r_a}{L_a} i_a - \frac{k_a}{L_a} \omega_r - \frac{1}{L_a} u_a$$

$$\frac{d\omega_r}{dt} = \frac{1}{J} k_a i_a - B_m \omega_r - T_L$$

and

$$\frac{d\theta_r}{dt} = \omega_r$$

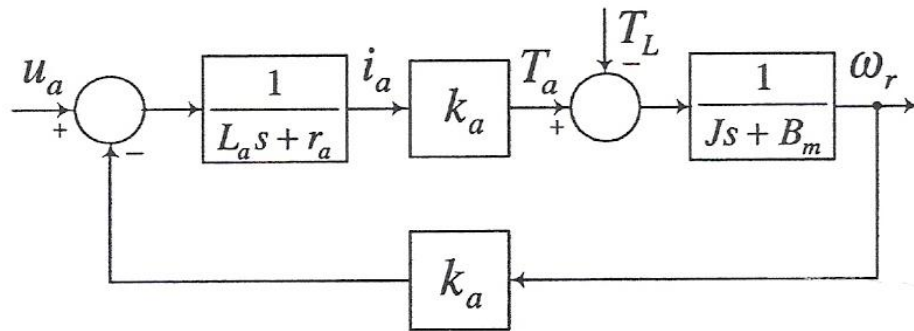
Using the Laplace operator  $s = \frac{d}{dt}$

$$\left(s + \frac{r_a}{L_a}\right) i_a(s) = -\frac{k_a}{L_a} \omega_r(s) + \frac{1}{L_a} u_a(s)$$

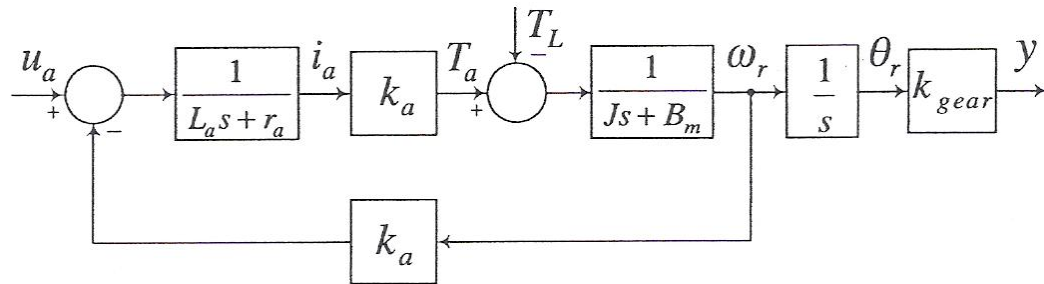
$$\left(s + \frac{B_m}{J}\right)\omega_r(s) = -\frac{1}{J}k_a i_a(s) + \frac{1}{J}T_L(s)$$

$$s\theta_r(s) = \omega_r(s)$$

From the transfer function, the block diagram of the permanent-magnet DC motor is illustrated by Figure 3.11



**Figure 3.11** Block Diagram of the Open-Loop Permanent-Magnet DC Motor



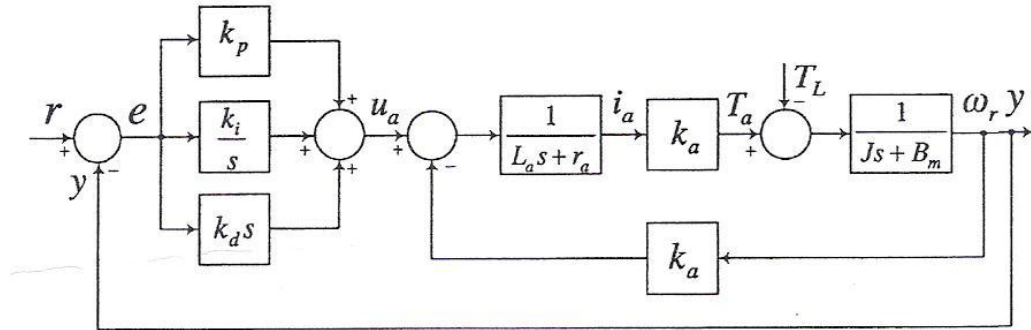
**Figure 3.12** Block Diagram of the Open-Loop Servo Actuated by Permanent-Magnet DC Motor

Using the linear PID controller

$$u_a(t) = k_p \theta(t) + k_i \frac{\theta(t)}{s} + k_d s \theta(t)$$



From the block diagram developed and documented in figure, it obtains the closed-loop system illustrated in Figure 3.13



**Figure 3.13** Block Diagram of the Closed-Loop Servo with PID Controller

In this project the permanent-magnet DC motor use is permanent-magnet Litton Clifton Precision JDH-2250-HF-2C-E. The parameters are:

$$\begin{aligned}
 R_a &= 2.7 \text{ ohm} & L_a &= 0.004 \text{ H} \\
 B_m &= 0.0000093 \text{ N-m-s-rad}^{-1} \\
 k_a &= 0.105 \text{ V-s-rad}^{-1} \text{ (the back emf constant)} \\
 k_a &= 0.105 \text{ N-m-A}^{-1} \text{ (the torque constant)} \\
 J &= 0.0001 \text{ kg-m}^2
 \end{aligned}$$

### 3.2 Data Acquisition

Data acquisition is the sampling of the real world in generating data that can be manipulated by a computer. Data acquisition typically involves acquisition of signals or waveforms then and processing the signals to obtain desired information. Components of data acquisition systems include sensors that convert any measurement parameter to an electrical signal, which is acquired by data acquisition hardware.

The acquired data from the data acquisition hardware are displayed, analyzed, and stored on a computer, either using software, or custom displays and control developed using programming languages such as BASIC, C, Fortran, Java, Lisp, Pascal. Programming languages that used for data acquisition include, EPICS, Lab VIEW, and MATLAB provides a programming language but also built-in graphical tools and libraries for data acquisition and analysis.

Transducer is a device that converts physical property or phenomenon into corresponding measurable electrical signal, such as voltage and current. The data acquisition system ability to measure different phenomena depends on the transducers to convert the physical phenomena into a signal measurable by the data acquisition hardware. There are specific transducers for many different applications, such as measuring temperature, pressure, or fluid flow. DAQ also deploy various Signal Conditioning techniques to adequately modify various different electrical signals into voltage that can then be digitized using ADCs.

Signals may be digital or analog depending on the transducer used. Signal conditioning may be necessary if the signal from the transducer is not suitable for the DAQ hardware that'll be used. The signal may be amplified or deamplified, or may require filtering, or a lock-in amplifier is included to perform demodulation. Various other examples of signal conditioning might be bridge completion, providing current or voltage excitation to the sensor, isolation, linearization, etc.

DAQ hardware is what usually interfaces between the signal and a PC. It could be in the form of modules that can be connected to the computer's ports (parallel, serial, USB, etc...) or cards connected to slots (PCI, ISA) in the mother board. Due to the space on the back of a PCI card is too small for all the connections needed, an external breakout box is required. DAQ-cards often contain multiple components (multiplexer, ADC, DAC, TTL-IO, high speed timers, RAM). These are accessible via a bus by a micro controller, which can run small programs. The controller is more flexible than a hard wired logic, yet cheaper than a CPU so that it is alright to block it with simple polling loops.

Driver software that usually comes with the DAQ hardware or from other vendors, allows the operating system to recognize the DAQ hardware and programs to access the signals being read by the DAQ hardware. A good driver offers high and low level access. So one would start out with the high level solutions offered and improves down to assembly instructions in time critical or exotic applications.

### **3.2.1 PCI-1710HG**

The Advantech PCI-1710HG is a powerful data acquisition (DAS) card for the PCI bus. It features a unique circuit design and complete functions for data acquisition and control, including A/D conversion, D/A conversion, digital input, digital output, and counter/timer. The Advantech PCI-1710HG provides users with the most requested measurement and control functions as below:

- PCI-bus mastering for data transfer
- 16-channel Single-Ended or 8 differential A/D Input
- 12-bit A/D conversion with up to 100 kHz sampling rate
- Programmable gain for each input channel
- On board samples FIFO buffer (4096 samples)
- 2-channel D/A Output
- 16-channel Digital Input
- 16-channel Digital Output
- Programmable Counter/Timer
- Automatic Channel/Gain Scanning
- Board ID

### 3.2.1.1 Specification

#### Analog Input

- **Channels** 16 single-ended/ 8 differential (SW programmable)
- **Resolution** 12 bits
- **Max. Sampling Rate\*** 100 kS/s
- **FIFO Size** 4096 samples
- **Overvoltage Protection**  $\pm 30\text{Vp-p}$
- **Input Impedance** 1 G $\Omega$
- **Sampling Modes** Software, onboard programmable pacer, or external
- **Input Range** (V, software programmable)

PCI-1710HG/1710HGL								
<b>Bipolar</b>	$\pm 10$	$\pm 5$	$\pm 1$	$\pm 0.5$	$\pm 0.1$	$\pm 0.05$	$\pm 0.01$	$\pm 0.005$
<b>Unipolar</b>	-	0 ~ 10	-	0 ~ 1	-	0 ~ 0.1	-	0 ~ 0.01
<b>Accuracy (% of FSR <math>\pm 1\text{LSB}</math>)</b>	0.1	0.1	0.2	0.2	0.2	0.2	0.4	0.4

**\*Note:**

The sampling rate and throughput depends on the computer hardware architecture and software environment. The rates may vary due to programming language, code efficiency, CPU utilization and so on.

#### Analog Output

- **Channels** 2
- **Resolution** 12 bits
- **Output Rate** Static update
- **Output Range** (V, software programmable)

<b>Internal Reference</b>	<b>Unipolar</b>	0 ~ +5 V @ -5 V 0 ~ +10 V @ -10 V
<b>External Reference</b>		0 ~ +x V @ -x V ( $-10 \leq x \leq 10$ )

- **Slew Rate** 10 V/ms
- **Driving Capability** 3 mA
- **Operation Mode** Software polling
- **Accuracy** INLE:  $\pm 1/2$  LSB, DNLE:  $\pm 1/2$  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: 8.0 mA @ 0.8 V  
Source: -0.4 mA @ 2.0 V

#### Pacer/Counter

- **Channels** 1
- **Resolution** 16 bits
- **Compatibility** 5 V/TTL
- **Max. Input Frequency** 1 MHz

#### General

- **Bus Type** PCI V2.2
- **I/O Connector** SCSI-68P female x 1
- **Dimensions (L x H)** 175 x 100 mm (6.9" x 3.9")

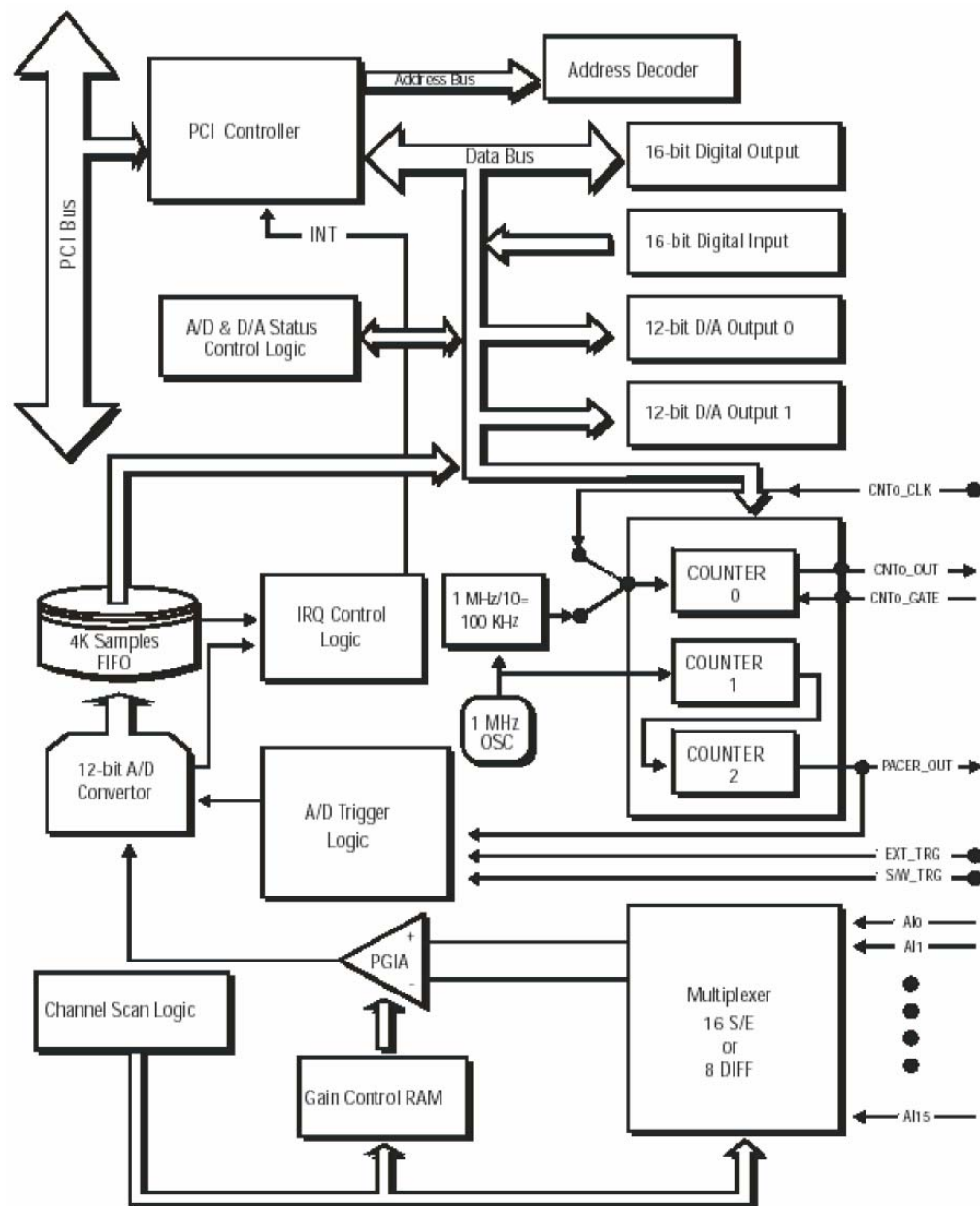
- **Power Consumption** Typical: 5 V @ 850 mA  
Max: 5 V @ 1.0 A
- **Operating Temperature** 0 ~ 60° C (32 ~ 140° F) (refer to IEC 68-2-1, 2)
- **Storing Temperature** -20 ~ 70° C (-4 ~ 158° F)
- **Storing Humidity** 5 ~ 95% RH non-condensing (refer to IEC 68-2-3)

### Pin Assignments

AI0	68	34	AI1
AI2	67	33	AI3
AI4	66	32	AI5
AI6	65	31	AI7
AI8	64	30	AI9
AI10	63	29	AI11
AI12	62	28	AI13
AI14	61	27	AI15
AIGND	60	26	AIGND
*AO0_REF	59	25	AO1_REF*
*AO0_OUT	58	24	AO1_OUT*
AOGND	57	23	AOGND
DI0	56	22	DI1
DI2	55	21	DI3
DI4	54	20	DI5
DI6	53	19	DI7
DI8	52	18	DI9
DI10	51	17	DI11
DI12	50	16	DI13
DI14	49	15	DI15
DGND	48	14	DGND
DO0	47	13	DO1
DO2	46	12	DO3
DO4	45	11	DO5
DO6	44	10	DO7
DO8	43	9	DO9
DO10	42	8	DO11
DO12	41	7	DO13
DO14	40	6	DO15
DGND	39	5	DGND
CNT0_CLK	38	4	PACER_OUT
CNT0_OUT	37	3	TRG_GATE
CNT0_GATE	36	2	EXT_TRG
+12V	35	1	+5V

\*: Pins 23~25 and pins 57~59 are not defined for PCI-1710L/1710HGL

**Figure 3.14** Pin Assignment



**Figure 3.15** Block Diagram of PCI-1710HG

### 3.2.1.2 Installation Guide

Before installing the PCI-1710HG card, make sure the following necessary component is present:

- **PCI-1710HG Multifunction card**
- **PCI-1710HG User's Manual**
- **Driver software** Advantech DLL drivers (included in the companion CD-ROM)
- **Wiring cable** PCL-10168
- **Wiring board** PCLD-8710, ADAM-3968
- **Computer** Personal computer or workstation with a PCI-bus slot (running *Windows95/98/NT/2000/XP*)

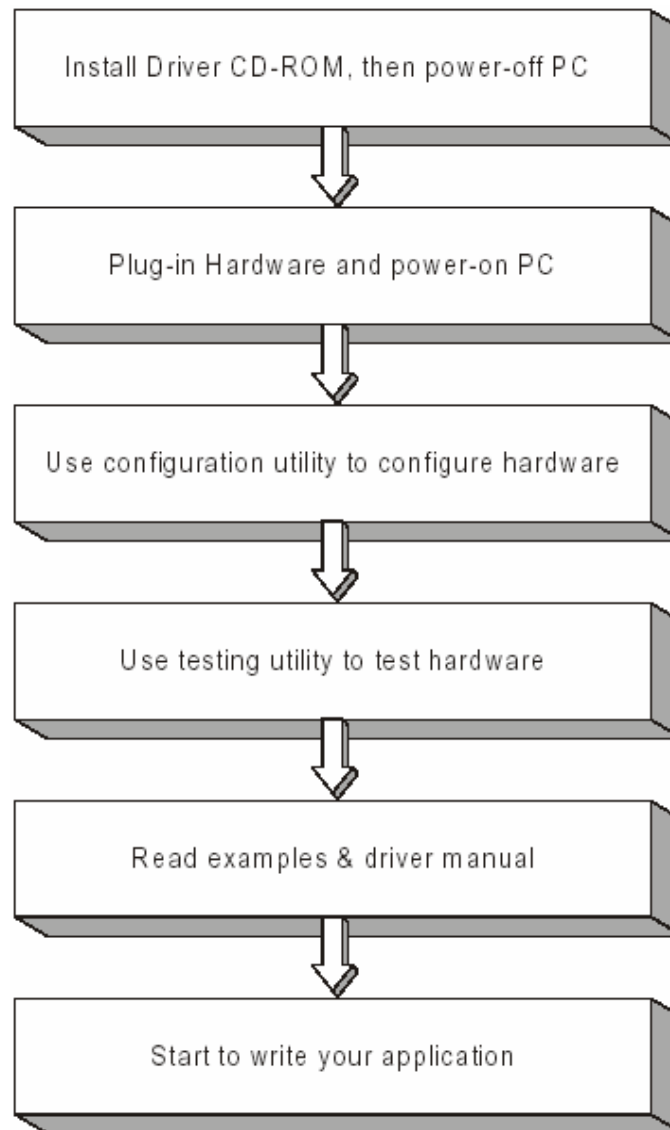
Some other optional components are also available for enhanced operation:

- **Application software** ActiveDAQ, GeniDAQ or other third-party software packages

After getting the necessary components and maybe some of the accessories for enhanced operation of the Multifunction card, begin the Installation procedures.

Figure 3.16 provides a concise flow chart to give a broad picture of the software and hardware installation procedures:





**Figure 3.16** PCI-1710HG Installation Flow Chart

Advantech offers a rich set of DLL drivers, third-party driver support and application software to help fully exploit the functions of the PCI-1710HG card:

- DLL driver (on the companion CD-ROM)
- LabVIEW driver
- Advantech ActiveDAQ
- Advantech GeniDAQ

### 3.3 Real Time Computing

Real-time computing is the study of hardware and software systems that are subject to a "real-time constraint" example, operational deadlines from event to system response. A *non-real-time system* is one for which there is no deadline, even if fast response or high performance is desired or even preferred. The needs of real-time software are often addressed in the context of real-time operating systems, and synchronous programming languages, which provide guide on which to build real-time application software.

A real time system may be one where its application can be considered to be mission critical. The anti-lock brakes on a car are an example of a real-time computing system, the real-time constraint in this system is the short time in which the brakes must be released to prevent the wheel from locking. Real-time computations can be said to have *failed* if they are not completed before their deadline, where their deadline is relative to an event. A real-time deadline must be met, regardless of system load.

The term real-time derives from its use in early simulation. While current usage implies that a computation that is 'fast enough' is real-time, originally it referred to a simulation that proceeded at a rate that matched that of the real process it was simulating. Analog computers, especially, were often capable of simulating much *faster* than real-time, a situation that could be just as dangerous as a slow simulation if it were not also recognized and accounted for.

Real-time computing is sometimes misunderstood to be high-performance computing, but this is not always the case. For example, a massive supercomputer executing a scientific simulation may offer impressive performance, yet it is not executing a real-time computation. Conversely, once the hardware and software for an anti-lock braking system has been designed to meet its required deadlines, no further performance gains are necessary. Furthermore, if a network server is highly loaded with network traffic, its response time may be slower but will still succeed.

Hence, such a network server would not be considered a real-time system, temporal failures (delays, time-outs, etc.) are typically small and compartmentalized (limited in effect) but are not catastrophic failures. In a real-time system, a slow-down beyond limits would often be considered catastrophic in its application context. Therefore, the most important requirement of a real time system is predictability and not performance.

Some kinds of software, such as many chess-playing programs, can fall into either category. For instance, a chess program designed to play in a tournament with a clock will need to decide on a move before a certain deadline or lose the game, and is therefore a real-time computation, but a chess program that is allowed to run indefinitely before moving is not. In both of these cases, however, high performance is desirable: the more work a tournament chess program can do in the allotted time, the better its moves will be, and the faster an unconstrained chess program runs, the sooner it will be able to move. This example also illustrates the essential difference between real-time computations and other computations, if the tournament chess program does not make a decision about its next move in its allotted time it loses the game, example if it fails as a real-time computation—while in the other scenario, meeting the deadline is assumed not to be necessary.

### **3.4 Real Time Window Target**

Real-Time Windows Target™ rapid prototyping software is a PC solution for prototyping and testing real-time systems. Real-Time Windows Target software uses a single computer as a host and target. On this computer, MATLAB® environment, Simulink® software, and Stateflow® software (optional) is use to create models using Simulink blocks and Stateflow diagrams.

After creating a model and simulating it using Simulink software in normal mode, it can generate executable code with Real-Time Workshop® code generation software, Stateflow® Coder™ code generation software (optional), and the Open Watcom C/C++ compiler. Then the application can be run in real time with Simulink external mode.

Real-Time Windows Target uses standard and inexpensive I/O boards for PC-compatible computers. When running the models in real time, Real-Time Windows Target captures the sampled data from one or more input channels, uses the data as inputs to the block diagram model, immediately processes the data, and sends it back to the outside world through an output channel on the I/O board.

Real-Time Windows Target provides a custom Simulink block library. The I/O driver block library contains universal drivers for supported I/O boards. These universal blocks are configured to operate with the library of supported drivers. This allows easy location of driver blocks and easy configuration of I/O boards.

It only need to drag and drop a universal I/O driver block from the I/O library the same way as it would from a standard Simulink block library. And it connects an I/O driver block to the model just as it would connect any standard Simulink block.

It just need to create a real-time application in the same way as it create any other Simulink model, by using standard blocks and C-code S-functions. It can add input and output devices to the Simulink model by using the I/O driver blocks from the rtwinlib library provided with Real-Time Windows Target. This library contains the following blocks:

- Analog Input
- Analog Output
- Counter Input
- Digital Input
- Digital Output
- Encoder Input

- Frequency Output
- Packet Input
- Packet Output
- Stream Input
- Stream Output

### **3.4.1 Setup and Configuration**

Real-time Window Target can use any PC compatible computer that runs Windows 2000, Windows XP 32-bit, or Windows Vista 32-bit. The computer can be a desktop, laptop, or notebook PC.

#### **3.4.1.1 Compiler**

Compiled code is created from the generated C-code using the Open Watcom C/C++ compiler. For convenience, this compiler is shipped with the Real-Time Windows Target software. No other third-party compiler is needed or can be used.

The Real-Time Windows Target software always uses the Open Watcom C/C++ compiler, even if you have specified some other compiler using the `mex -setup` command. Real-Time Windows Target software cannot be configured to use a compiler other than Open Watcom C/C++.

### 3.4.1.2 Kernel Setup

During software installation, all Real-Time Windows Target software is copied onto the hard drive, but the Real-Time Windows Target kernel is not automatically installed into the operating system. The kernel must be installed before a Real-Time Windows Target application can be run. Installing the kernel configures it to start running in the background each time the computer is start. The following procedure describes how to use the command `rtwintgt -install`. The command `rtwintgt -setup` can also be used instead. To install the kernel:

1. `rtwintgt -install`

is typed in the MATLAB Command Window

or:

- a) Click the MATLAB **Start** button.
- b) Select **Links and Targets > Real-Time Windows Target > Install real-time kernel**

The MATLAB Command Window displays one of these messages:

You are going to install the Real-Time Windows Target kernel.

Do you want to proceed? [y] :

**or:**

There is a different version of the Real-Time Windows Target kernel installed.

Do you want to update to the current version? [y] :

2. `y` is typed to continue installing the kernel, or `n` to cancel installation without making any changes.

If `y`, the MATLAB environment installs the kernel and displays the message:

```
The Real-Time Windows Target kernel has been
successfully installed.
```

3. If a message appears asking to restart your computer, do so before attempting to use the kernel, or the Real-Time Windows Target model will not run correctly.
4. After installing the kernel, verify that it was correctly installed by typing:

```
rtwho
```

The MATLAB Command Window should display a message that shows the kernel version number, followed by performance, timeslice, and other information similar to bellow

```
Real-Time Windows Target version 3.0.0 (C) The
MathWorks, Inc. 1994-2007
Running on Uniprocessor APIC computer.
MATLAB performance = 100.0%
Kernel timeslice period = 1 ms
```

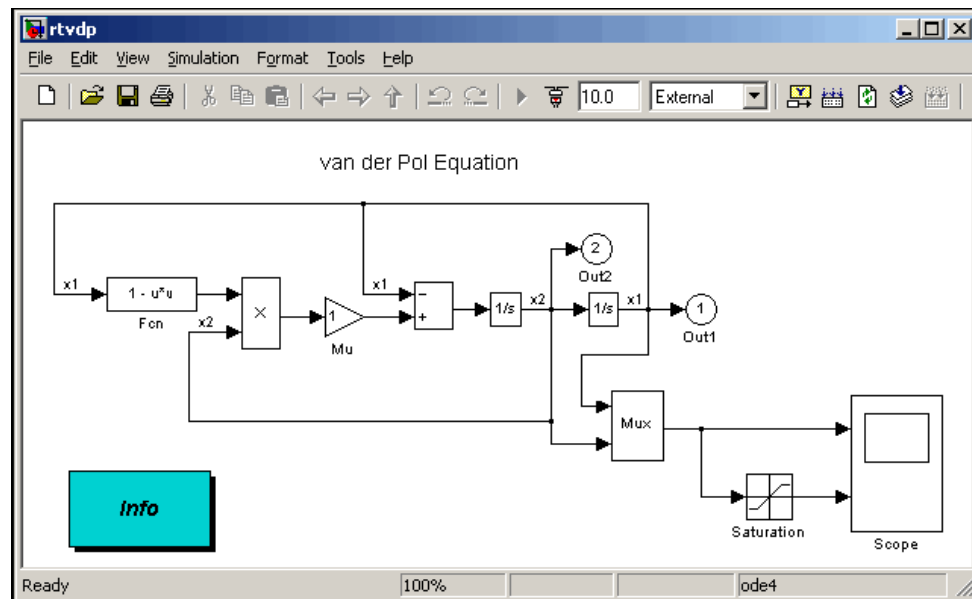
Once the kernel is installed, you can leave it installed. The kernel remains idle after you have installed it, which allows the Windows operating system to control the execution of any standard Windows based application, including Internet browsers, word processors, the MATLAB environment, and so on. The kernel becomes active when you begin execution of your model, and becomes idle again after model execution completes.

### 3.4.1.3 Testing the Installation

Once the installation of the Real-Time Windows Target software and kernel is completed, it is recommended a quick test by running the model `rtvdp.mdl`. Doing this test is a quick check to confirm that the Real-Time Windows Target software is still working. The model `rtvdp.mdl` does not have any I/O blocks, so that this model can run regardless of the I/O boards in your computer. Running this model will test the installation by running Real-Time Workshop code generation software, Real-Time Windows Target software, and the Real-Time Windows Target kernel.

1. To open the demo model `rtvdp` is typed in the MATLAB Command Window, or launch MATLAB Online Help and choose **Demos > Links and Targets > Real-Time Windows Target > Real-Time Van der Pol Simulation**.

The Simulink model `rtvdp.mdl` window opens.



**Figure 3.17** Simulink Model `rtvdp.mdl`



2. From the **Tools** menu, **Real-Time Workshop** is choose and **Build Model** is selected. The MATLAB Command Window displays the following messages:

```
### Starting Real-Time Workshop build for model: rtvdp
### Invoking Target Language Compiler on rtvdp.rtw
. . .
### Compiling rtvdp.c
. . .
### Created Real-Time Windows Target module rtvdp.rwd.
### Successful completion of Real-Time Workshop build
procedure
for model: rtvdp
```

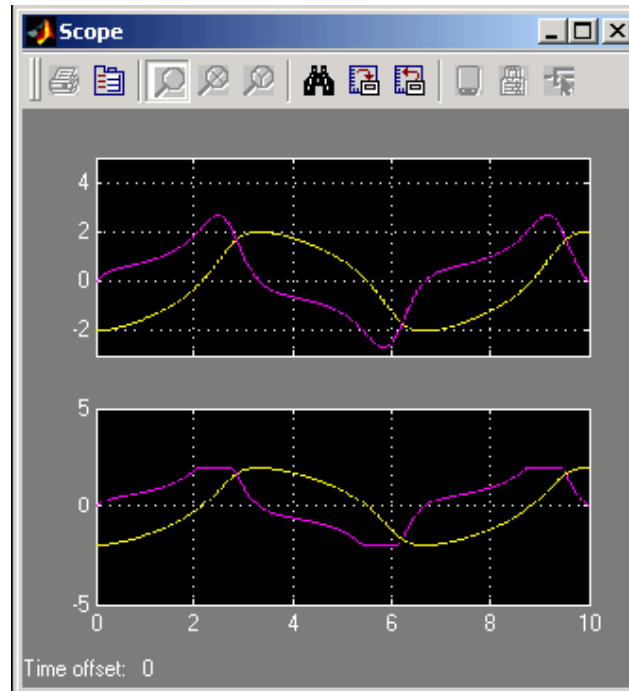
3. From the **Simulation** menu, click **External**, and then click **Connect to target**.

The MATLAB Command Window displays the following message:

```
Model rtvdp loaded
```

4. From **Simulation** menu, **Start Real-Time Code** is clicked.

The Scope window displays the output signals. If the Scope window looks like the next figure, the Real-Time Windows Target software have successfully installed and have run a real-time application.



**Figure 3.18** Output Signal rtvdp.mdl

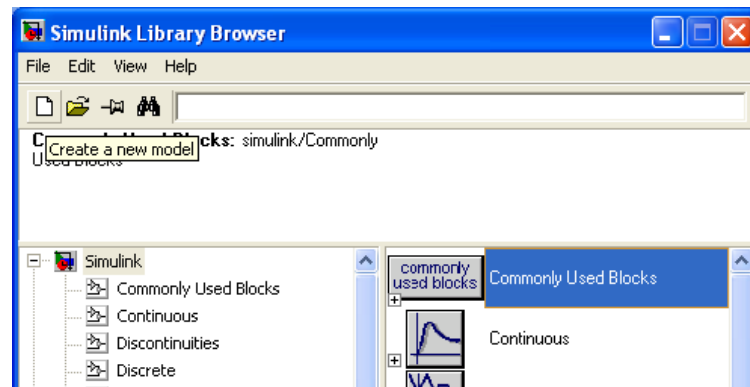
5. From **Simulation** menu, **Stop Real-Time Code** is selected. The real-time application stops running, and the Scope window stops displaying the output signals.

### 3.4.2 Creating a Real Time Application

This procedure explains how to create a simple Simulink model. This model is used as an example to learn other procedures that are useful with Real-Time Windows Target software. A Simulink model is created before running a simulation or creates a Real-Time Target software

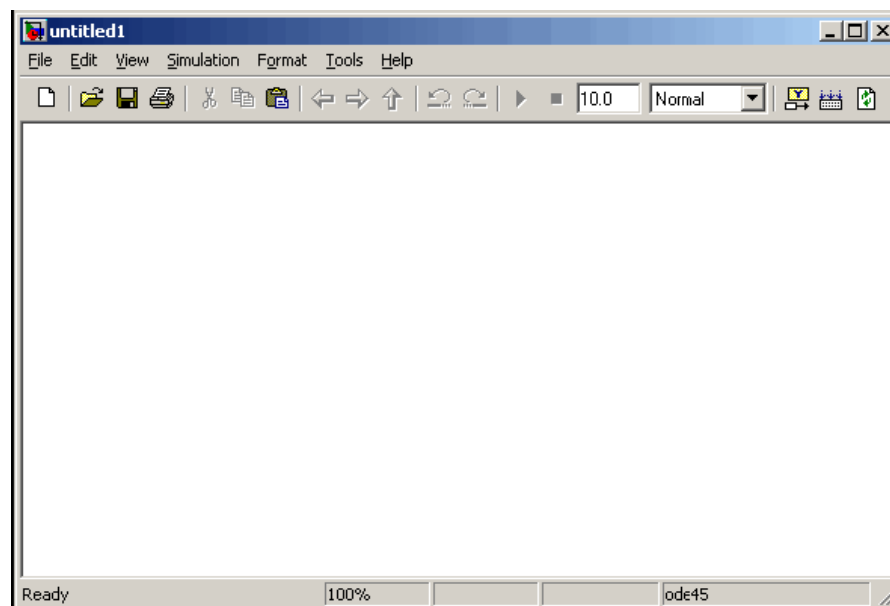
1. In the MATLAB Command Window, `simulink` is typed

The Simulink Library Browser opens. The left pane shows a hierarchy of libraries and block categories, with the Simulink library at the top. The right pane shows the blocks available in the category selected on the left. See “Library Browser” for more information.



**Figure 3.19** Simulink Library Browser

2. Choose **File > New>Model**, or the **New model** button is click on the toolbar.  
An empty Simulink window opens:



**Figure 3.20** Empty Simulink Windows

3. In the left pane of the Simulink Library Browser window, **Simulink** is click and **Sources** is choose. Then **Signal Generator** block is click and drag from the browser to the Simulink window. From **Sinks**, **Scope** is click and drag to the Simulink window. Real-Time Window Target is selected and **Analog Output** is drag to the Simulink window.

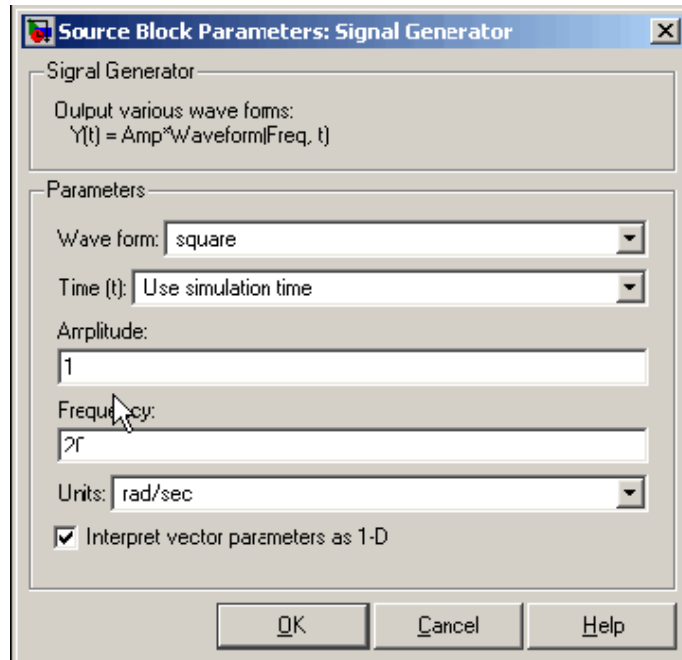
4. The **Signal Generator** output is connected to the **Scope** input by clicking-and-dragging a line between the blocks. Likewise, the **Analog Output** is connected between the **Scope** and **Signal Generator**.
5. Signal Generator block is double clicked. The Block Parameters dialog box opens. From the **Wave form** list, square is choose.

In the **Amplitude** text box, 1 is entered

In the **Frequency** text box, 20 is entered

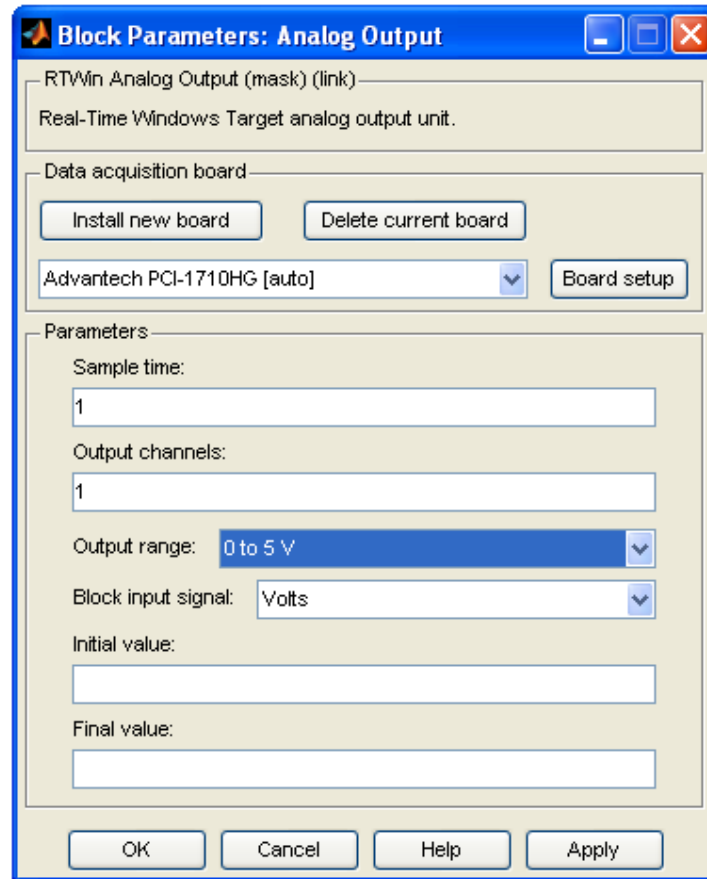
From the **Units** list, rad/sec is selected.

The Block Parameters dialog box shown in Figure 3.21.



**Figure 3.21** Signal Generator Block Parameter

6. **OK** is clicked.
7. The **Analog Output** block is double clicked.  
The **Analog Output** Block Parameters dialog box opens. The I/O Block parameters dialog box opens. For an Analog Output block, the dialog box is shown in Figure 3.22.



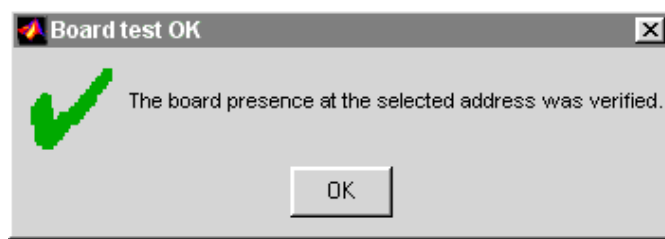
**Figure 3.22** Analog Output Block Parameter

8. **Install new board** is clicked. From the list that appears, the manufacturer of the board is pointed, and then a board type is selected. For example, it pointed to **Advantech**, then click **PCI-1710HG**.
9. One of the following is selected, as appropriate to the board:
  - For an ISA bus board, a hexadecimal base address is entered. This value must match the base address jumpers or switches set on the physical board. For example, if a base address of 0x300 is entered, in the **Address** box 300 typed. The base address is selected by checking boxes **A9** through **A3**.
  - For a PCI bus board, the **PCI slot is** entered or **Auto-detect** are checked.

10. The Block Parameters dialog also able to set other block parameters, such as the sample time. Set such parameters as needed.

11. Then **Test** is clicked.

The Real-Time Windows Target kernel tries to connect to the selected board, and if successful, displays the following message.



**Figure 3.23** Board Test OK Dialog

12. **OK** button on the message box is clicked, and again on the Block Parameters dialog box.

The I/O Block Parameters dialog box closes, and the parameter values are included in your Simulink model.

13. In the **Sample time** box, enter the same value entered in the **Fixed step** size box from the Configuration Parameters dialog box, or an integer multiple of that value 0.001

14. In the **Output channels** box, a channel vector is entered that selects the analog output channels used on this board. The vector can be any valid MATLAB vector form. For example, to select single analog output channels on the **PCI-1710HG** board, 1 is entered or to select both analog output channels, [1,2] or [1:2] is entered

15. For the Output range list, the input range for all of the analog input channels entered in the Input channels box is copied. For example, with the **PCI-1710HG** board, 0 to 10 V is choose.

16. From the Block input signal list, choose from the following options:

- Volts — Expects a value equal to the analog output voltage.
- Normalized unipolar — Expects a value between 0 and +1 that is converted to the full range of the output voltage regardless of the output voltage range. For example, an analog output range of 0 to +5 volts and -5 to +5 volts would both be converted from values between 0 and +1.
- Normalized bipolar — Expects a value between -1 and +1 that is converted to the full range of the output voltage regardless of the output voltage range.
- Raw — Expects a value of 0 to  $2^n - 1$ . For example, a 12-bit A/D converter would expect a value between 0 and  $2^{12} - 1$  (0 to 4095). The advantage of this method is the expected value is always an integer with no round off errors.

17. The initial value for each analog **Output channel** entered in the Output channels box. For example, if 1 was entered in the **Output channels** box, and an initial value of 0 volts is needed, enter 0.

18. The final value is entered for each analog channel entered in the **Output channels** box. For example, if 1 is entered in the **Output channels** box, and a final value of 0 volts is needed, 0 is entered.

19. One of the following is clicked:

- **Apply** to apply the changes to your model and leave the dialog box open.
- **OK** to apply the changes to your model and close the dialog box.

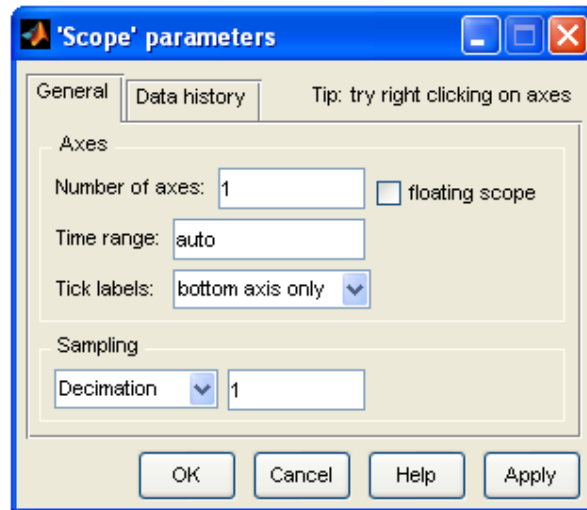
20. In the Simulink window, the Scope block is double click.

A Scope window opens.

21. The Parameters button is click.

A Scope parameters dialog box opens.

22. The **General** tab is clicked. In the **Number of axes** field, the number of graphs needed in one Scope window is entered. For example, 1 is entered for a single graph. Do not select the floating scope check box. In the **Time range** field, the upper value for the time range is entered. For example, 1 second is entered. From the **Tick labels** list, **bottom axis only** is chosen. From the **Sampling** list, **decimation** is chosen and 1 is entered in the text box. The Scope parameters dialog box looks similar to Figure 3.24.



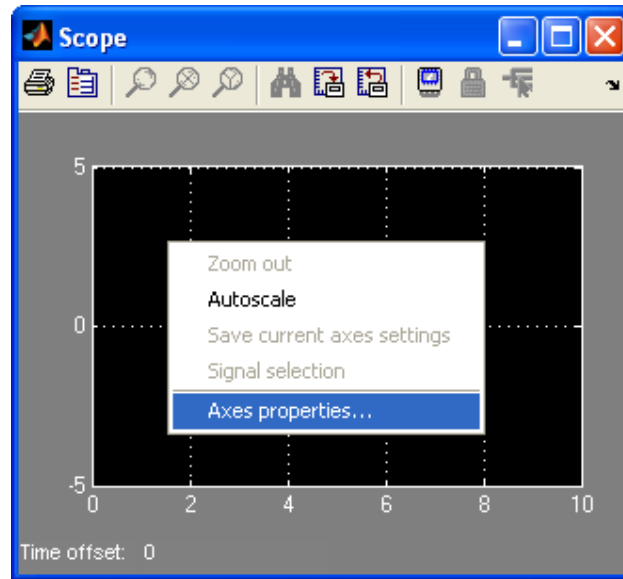
**Figure 3.24** Scope Parameters Dialog Box

23. One of the following is clicked:

- **Apply** to apply the changes to your model and leave the dialog box open.
- **OK** to apply the changes to your model and close the dialog box.

24. In the Scope window, the y-axis is right-clicked. From the menu, **Axes Properties** is clicked.

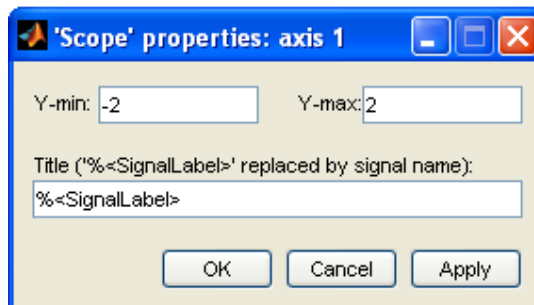




**Figure 3.25** Scope Window

The Scope properties: axis 1 dialog box opens.

25. In the **Y-min** and **Y-max** text boxes the range for the y-axis is entered in the Scope window. For example, -2 and 2 is entered as in Figure 3.26.

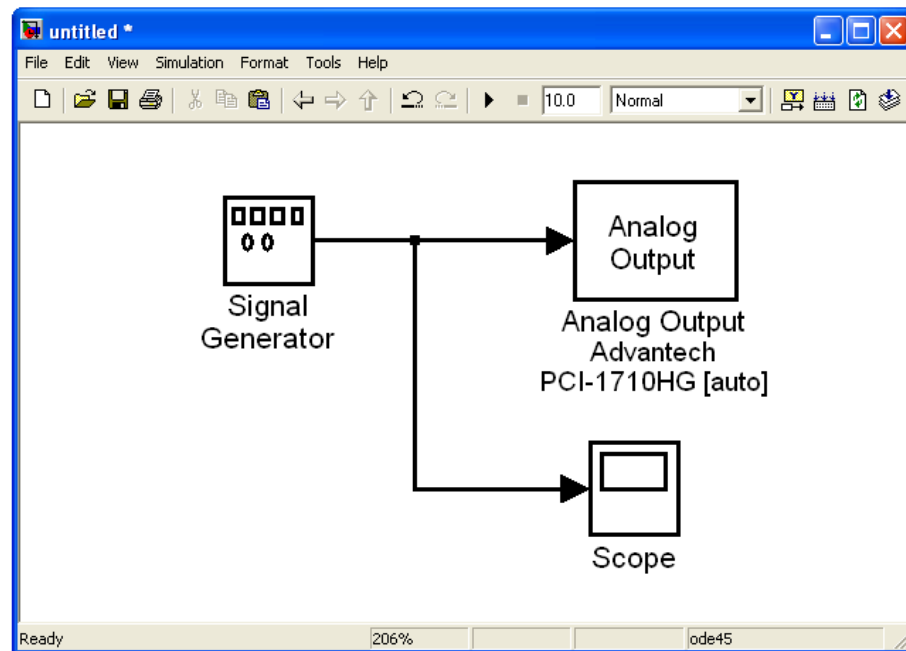


**Figure 3.26** Scope Properties: axis 1

26. One of the following is clicked:

- **Apply** to apply the changes to your model and leave the dialog box open.
- **OK** to apply the changes to your model and close the dialog box.

The complete Simulink block diagram is shown in Figure 3.27.



**Figure 3.27** Completed Simulink Block Diagram

27. From the **File** menu, **Save As** is clicked. The Save As dialog box opens. In the **File name** text box, a filename for the Simulink model is entered and **Save** is clicked. For example, `rtwin_model` is typed. The Simulink software saves your model in the file `rtwin_model.mdl`.

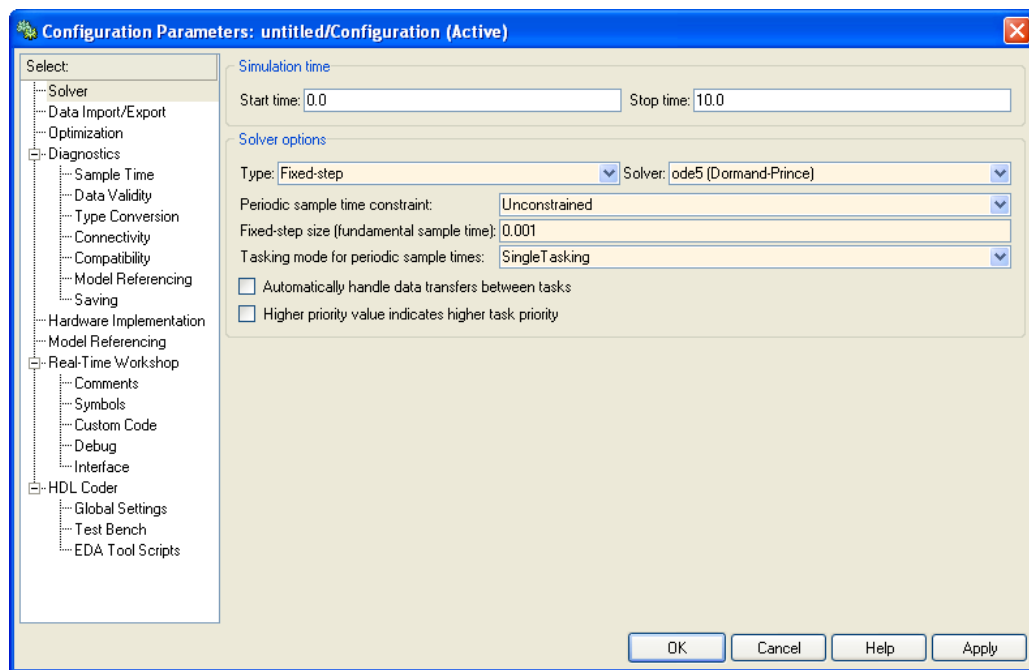
### 3.4.3 Entering Configuration Parameters for Simulink

The configuration parameters give information to Simulink for running a simulation. After creating a Simulink model, enter the configuration parameters for Simulink. This procedure uses the Simulink model `rtwin_model.mdl` as an example and assumes that the model has already been loaded.

1. In the Simulink window, and from the **Simulation** menu, **Configuration Parameters** is clicked. In the Configuration Parameters dialog box, the **Solver** tab is clicked. The **Solver** pane opens.

2. In the **Start time** field, 0.0 is entered. In the **Stop time** field, the amount of time the model needs run is entered. For example, 10.0 seconds is entered.
3. From the **Type** list, Fixed-step is chosen. Real-Time Workshop does not support variable step solvers.
4. From the **Solver** list, a solver is chosen. For example, the general purpose solver ode5 (Dormand-Prince) is chosen.
5. In the **Fixed step size** field, a sample time is entered. For example, 0.001 seconds is entered for a sample rate of 1000 samples/second.
6. From the **Tasking Mode** list, SingleTasking is picked. (For models with blocks that have different sample times, MultiTasking is picked.)

Your **Solver** pane looks similar to the next Figure 3.28.



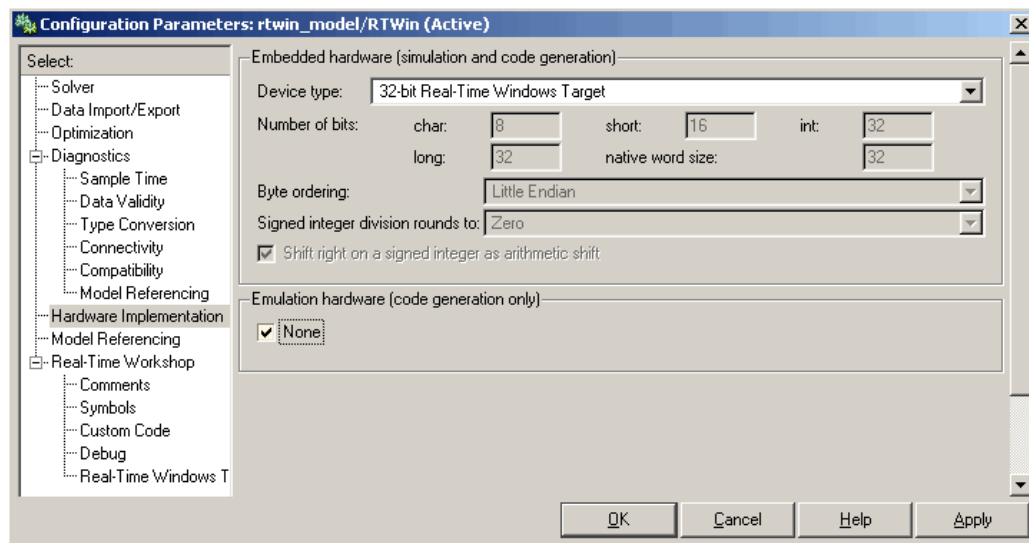
**Figure 3.28** Configuration Parameter (Solver) Windows.

7. One of the following is clicked:
  - **Apply** to apply the changes to your model and leave the dialog box open.
  - **OK** to apply the changes to your model and close the dialog box.

### 3.4.4 Entering Simulation Parameters for Real-Time Workshop

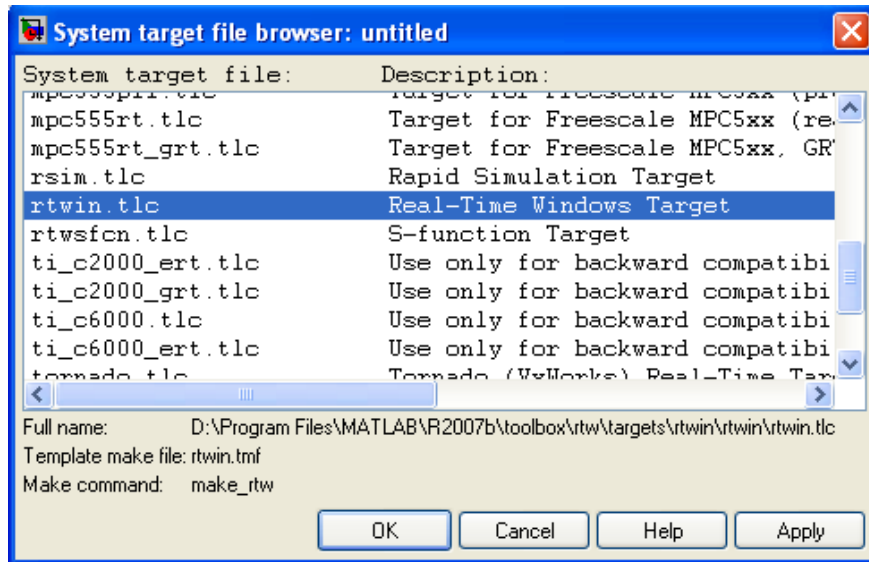
After creating a Simulink model, the simulation parameters for Real-Time Workshop is could be entered. The simulation parameters are used by Real-Time workshop for generating C code and building a real-time application. This procedure uses the Simulink model `rtwin_model.mdl` as an example and assumes it have already loaded that model:

1. In the Simulink window, and from the Simulation menu, **Configuration Parameters** is clicked and a window as in Figure 3.29 appears.



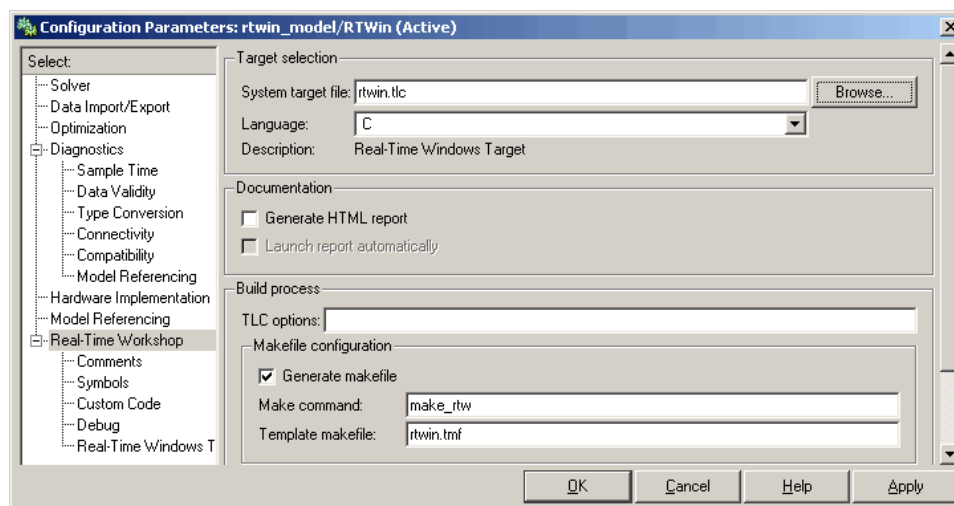
**Figure 3.29** Configuration Parameter (Hardware Implementation) Windows.

2. The **Hardware Implementation** node is click.
3. From the **Device type** list, 32-bit Real-Time Windows Target is chosen. Under **Emulation hardware**, none is selected.
4. The **Real-Time Workshop** node is clicked. The Real-Time Workshop pane opens.
5. In the **Target selection** section, the **Browse** button at the **RTW system target** file list is clicked as shown on Figure 3.30.
6. The System Target File Browser opens. The system target file for Real-Time Windows Target is selected and OK is clicked.



**Figure 3.30** System Target File Browsers.

The system target file `rtwin.tlc`, the template make file `rtwin.tmf`, and the make command `make_rtw` are automatically entered into the **Real-Time Workshop** pane. Although not visible in the **Real-Time Workshop** pane, the external target interface MEX file `rtwinext` is also configured when OK is clicked. This allows external mode to pass new parameters to the real-time application and to return signal data from the real-time application. The data is displayed in Scope blocks or saved with signal logging. The Real-Time Workshop pane looks similar to the Figure 3.31.



**Figure 3.31** Configuration Parameter (Real-Time Workshop) Windows.

7. One of the following is clicked:
  - **Apply** to apply the changes to your model and leave the dialog box open.
  - **OK** to apply the changes to your model and close the dialog box.

### 3.4.5 Creating a Real-Time Application

Real-Time Workshop generates C code from your Simulink model, then the **Microsoft Visual C++** compiler compiles and links that C code into a real-time application. After you enter parameters into the **Configuration Parameters** dialog box for Real-Time Workshop, a real-time application can be build. This procedure uses the Simulink model `rtwin_model.mdl` as an example, and assumes the model is loaded:

1. In the Simulink window, and from the **Tools** menu, point to **Real-Time Workshop**, and **Build Model** is clicked. The build process does the following:
  - Real-Time Workshop creates the C code source files `rtwin_model.c` and `rtwin_model.h`.
  - The make utility `make_rtw.exe` creates the make file `rtwin_model.mk` from the template make file `rtwin.tmf`.
  - The make utility `make_rtw.exe` builds the real-time application `rtwin_model.rwd` using the make file `rtwin_model.mk` created above. The file `rtwin_model.rwd` is a binary file that we refer to as your real-time application. You can run the real-time application with the Real-Time Windows Target kernel.
2. The Simulink model is connected to the real-time application .After creating a real-time application, MATLAB is closed and restarted, then it is connected and the executable is run without having to rebuild.


### 3.4.6 Running a Real-Time Application

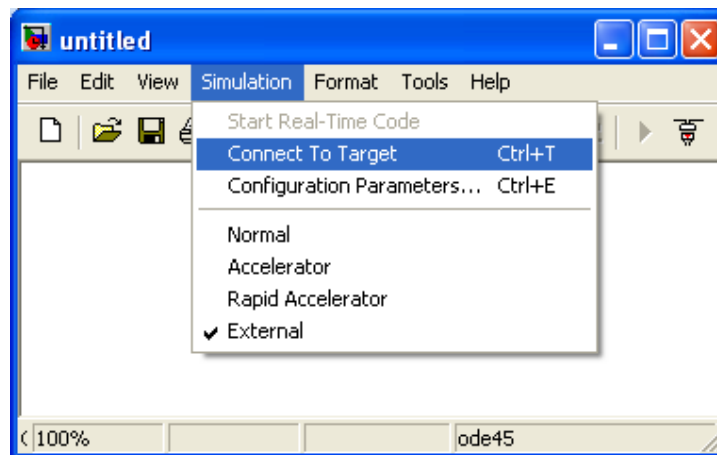
The real-time application is run to observe the behavior of your model in real time with the generated code.

The process of connecting consists of

- Establishing a connection between the Simulink model and the kernel to allow exchange of commands, parameters, and logged data.
- Running the application in real time.

After building the real-time application, the model is run in real time. This procedure uses the Simulink model `rtwin_model.mdl` as an example, and assumes the real-time application is created for that model:


1. From the Simulation menu, **External** mode simulation is selected. **Connect to Target** is chosen or clicking connects to the target from the toolbar  as in Figure 3.32.



**Figure 3.32** Connect To Target and Start Real-Time Code

MATLAB displays the message

```
Model rtwin_model loaded
```

2. From the Simulation menu, **Start Real-Time Code** is chosen or clicking start the code from the toolbar  as in figure 3.18.

Simulink runs the execution and plots the signal data in the Scope window. In this example, the Scope window displays 1000 samples in 1 second, increases the time offset, and then displays the samples for the next 1 second.

**Note:**

*Transfer of data is less critical than calculating the signal outputs at the selected sample interval. Therefore, data transfer runs at a lower priority in the remaining CPU time after real-time application computations are performed while waiting for another interrupt to trigger the next real-time application update. The result may be a loss of data points displayed in the Scope window.*

3. One of the following is done:
  - The execution run until it reaches the stop time.
  - From the Simulation menu, **Stop Real-time Code** is clicked.

The real-time application stops.
4. In the Simulation window, and from the Simulation menu, **Disconnect From Target** is clicked.
5. From the **Simulation** menu, **External** is clicked.

MATLAB displays the message

```
Model rtwin_model unloaded
```

### 3.5 Driver

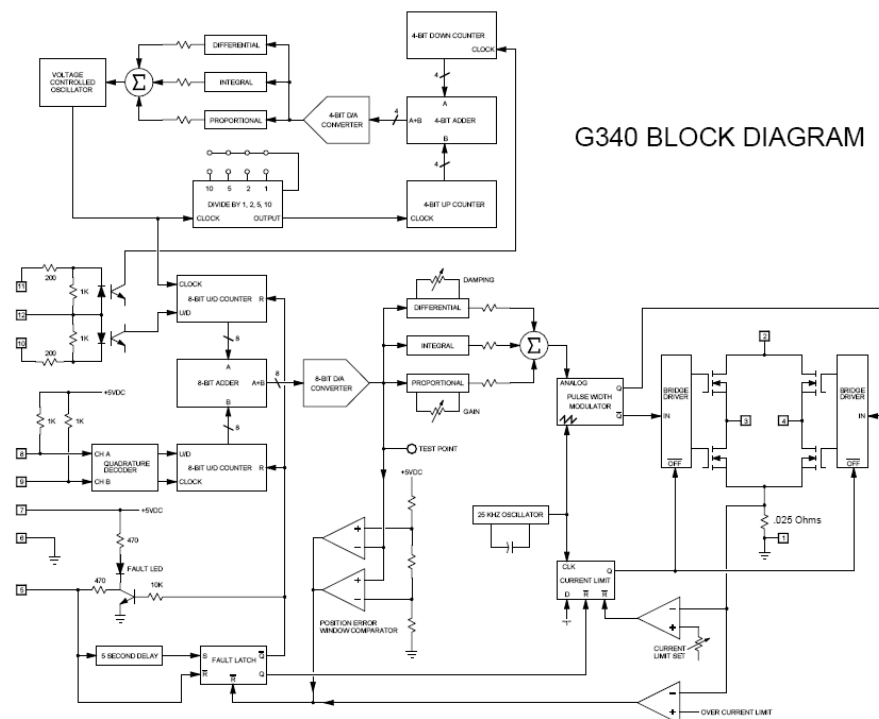
A driver is an electronic circuit which enables a voltage to be applied across a load in either direction. These circuits are often used in robotics and other applications to allow DC motors to run forwards or backwards. Driver available as integrated circuits, or can be built from discrete components. The reason why a driver is used to be connected to the motor is because the in capability of the DAQ Card to supply voltage higher than 10V. The DC motor used needed to be supplied



with 30 Volt in order to operate smoothly. With the driver it makes it possible to connect and external power supply to the motor by controlling it trough the driver.

### 3.5.1 Geckodrive G340

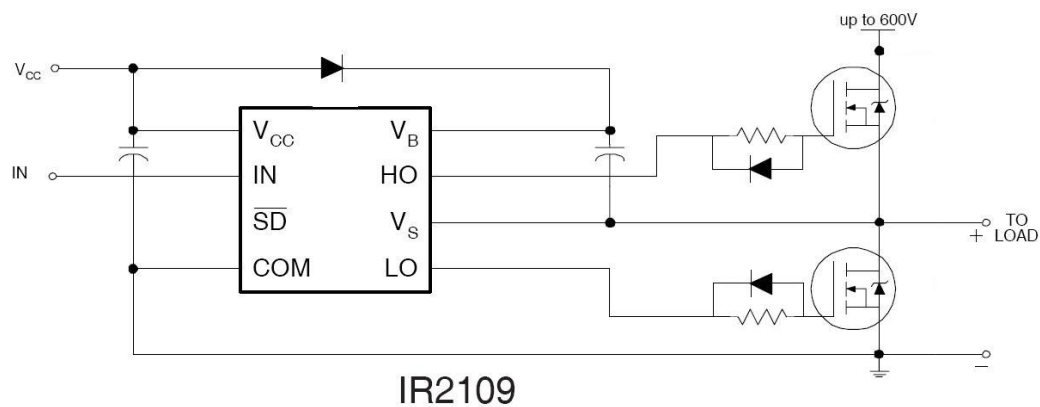
G340 is a PID feedback servo drive that could be used with motor rate up to 80Volt and 20Amp. The driver provide quadrature encoder inputs to be feedback to the PID controller inside the driver, the driver also provide 5Volt 50Mamp encoder supply for the motor encoder. Opto-isolated step and direction inputs are available for position, speed and direction control. Build in the driver is a 20 kHz PWM generator and a adjustable current limiter for protection. Due to it advance feature and especially the build in PID controller as in Figure 3.33, it was not compatible to be used it in this project.



**Figure 3.33** Geckodrive G340 Block Diagram

### 3.5.2 Alternative Driver IR2109

The IR2109 are high voltage, high speed power MOSFET and IGBT drivers with dependent high and low side referenced output channels. Proprietary HVIC and latch immune CMOS technologies enable ruggedized monolithic construction. The logic input is compatible with standard CMOS or LSTTL output, down to 3.3V logic. The output drivers feature a high pulse current buffer stage designed for minimum driver cross-conduction. The floating channel can be used to drive an N-channel power MOSFET or IGBT in the high side configuration which operates up to 600 volts. This driver. Figure 3.34 are the typical circuit connection for the driver.

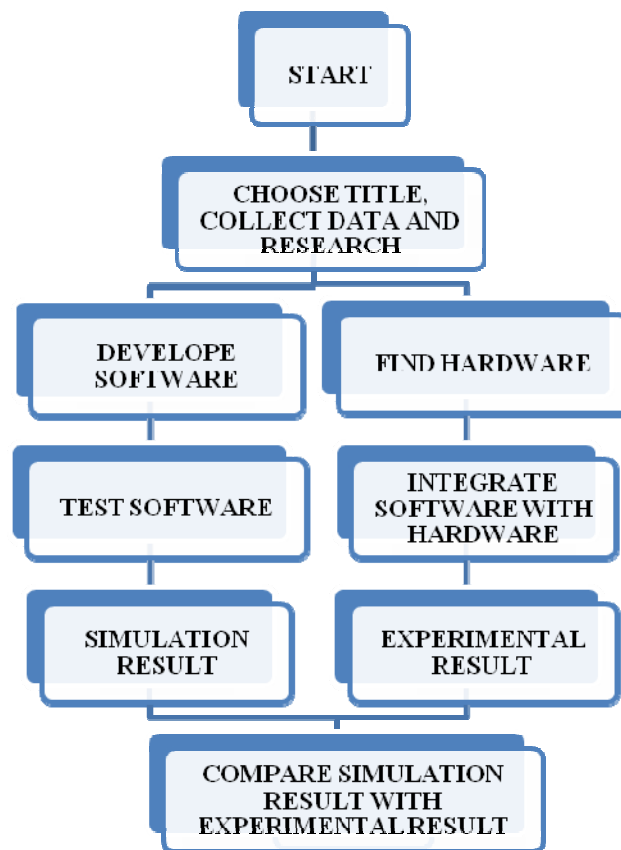


**Figure 3.34** Typical Connections for IR2109

The driver operates when a pulse or a PWM is send to the IN terminal of the driver, this will control the frequency of the two MOSFET gate to pass the voltage to the motor. The higher the width of the on cycle of the pulse the more voltage is passed to the motor therefore increasing the speed of the motor.

### 3.6 Project Planning

Figure 3.35 shows the flow chart of the development of the simulation and experimental of PID controller design for controlling DC motor speed using MATLAB application.



**Figure 3.35** Flow Chart of Project

## **CHAPTER 4**

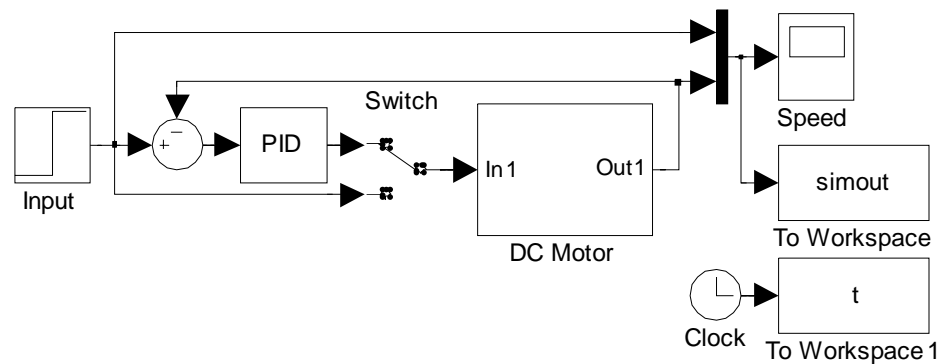
### **RESULT AND DISCUSSION**

#### **4.1 Controller Design**

In this experiment, PID controller was proposed to control the DC motor speed. The purpose of this part is to show how settings for controllers can be obtained from knowledge of the process to be controlled. This forms part of the complete control system design procedure. After manipulated and adjusted quantities have been selected and their pairings, perhaps tentatively, chosen then values of one or more parameters for each controller must be determined. The process with these control loops and controller settings can then be tested, usually by simulation using a mathematical model of the process and then with the actual process. The choice of control loops and/or the controller settings may then be changed if their performance is not satisfactory.

### 4.1.1 PID Controller

PID controller is the most widely use controller in the industrial control system. To design a PID controller there several method that could be used. The reason why PID controller were choose for this project is because that it get the desired output in a short time, with minimal overshoot and little error and also it is relatively easy to be implement. Figure 4.1 show a Simulink block of the PID controller.



**Figure 4.1** Simulink Block of PID Controller

#### 4.1.1.1 Zeigler Nichols Method

In 1942, John G. Ziegler and Nathaniel B. Nichols of Taylor Instruments published a paper on closed loop-tuning techniques that remain popular to this day. Ziegler and Nichols described a closed loop-tuning technique that is conducted with the controller in automatic mode, but with the integral and derivative actions set to zero. The Proportional gain is increased until even the slightest error causes a sustained oscillation in the process variable.

The smallest controller gain that can cause such an oscillation is called the ultimate gain  $K_{pmax}$ . The period of those oscillations is called the ultimate period  $T_{osc}$ .

The appropriate tuning parameters can be computed from these two values substituting it in the Table 4.1.

**Table 4.1:** Typical Values of Proportional, Integral, and Derivative feedback Coefficient for PID-type Controller

Controller	$K_p$	$K_i$	$K_d$
P	$0.5 K_{pmax}$	-	-
PI	$0.45 K_{pmax}$	$1.2 T_{osc}$	-
PID	$0.6 K_{pmax}$	$2 T_{osc}$	$T_{osc} / 8$

From the table, the value of  $K_p$ ,  $K_i$ , and  $K_d$  is applied to into the system.

#### 4.1.1.2 Trial and Error Method

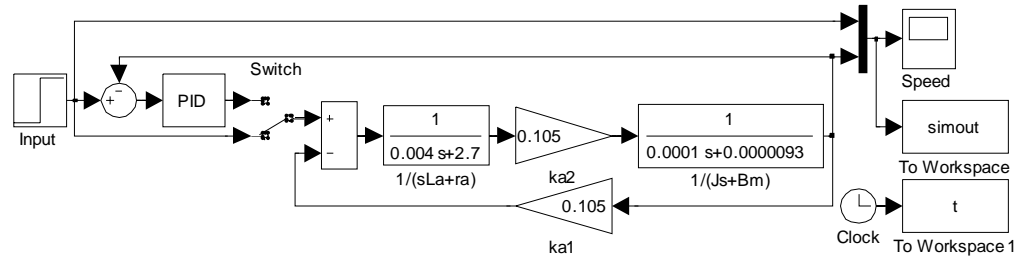
Trial and error, or trial by error, is a general method of problem solving for obtaining knowledge, both propositional knowledge and know-how. In the field of computer science, the method is called generate and test. In elementary algebra, when solving equations, it is "guess and check". This approach can be seen as one of the two basic approaches to problem solving and is contrasted with an approach using insight and theory.

Due to the unsuccessful use of Ziegler Nichols method where when  $K_p$  was increase the step pulse didn't oscillate. Therefore to find the value of  $K_p$ ,  $K_i$ , and  $K_d$ , trial and error method was the resort. Trough this method the value of  $K_p$ ,  $K_i$ , and  $K_d$ , was obtain by increasing their value until the best result are obtain. In this simulation the values obtain is as follow:

$$K_p = 15 \quad K_i = 86 \quad K_d = 0.05$$

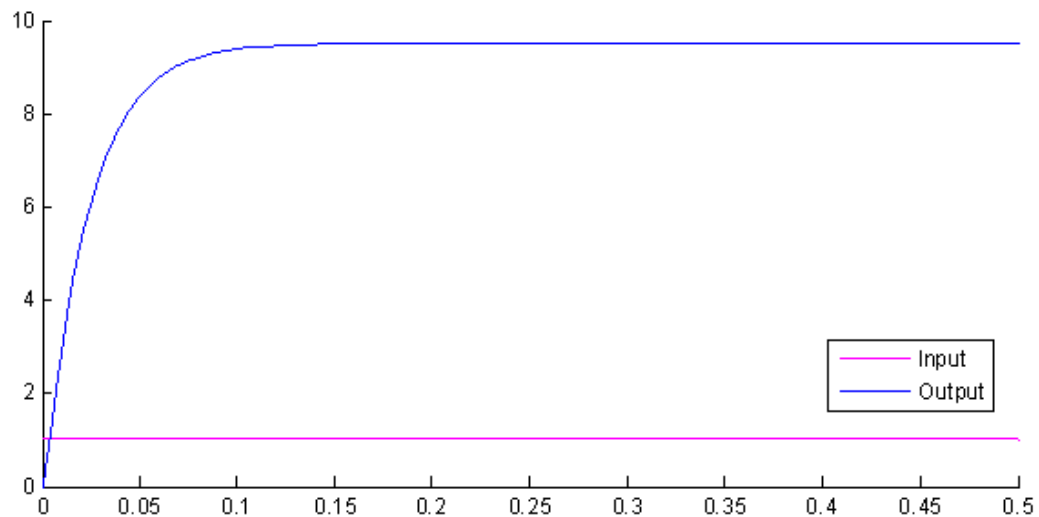
## 4.2 Simulation without PID Controller

The detailed and explicit simulation block for the DC motor without PID controller is shown in Figure 4.2.



**Figure 4.2** Detailed Simulink Block of the System

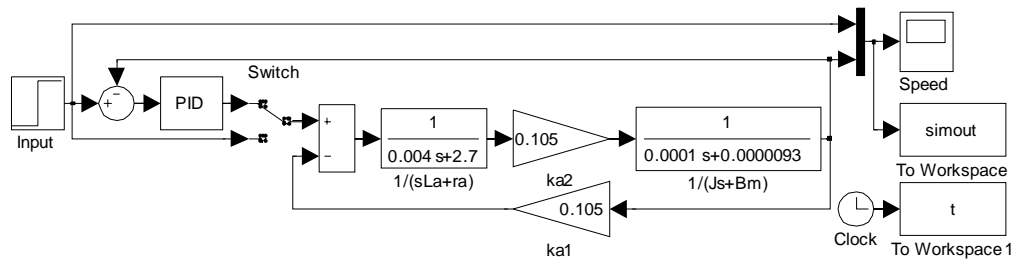
The DC motor parameter  $r_a$ ,  $L_a$ ,  $B_m$ ,  $k_a$ , and  $J$  is entered in the simulation block. Assigning the desired reference speed of the DC motor to be  $r(t) = 1$  rad, the modeling was performed and Figure 4.2 illustrates the output dynamic. As it can be seen there is a large error from the output show from the simulation. The steady-state error is 8.503 rad.



**Figure 4.3** Output of DC Motor without PID Controller

### 4.3 Simulation with PID Controller

After simulating the system without PID controller, PID controller is implemented to the Simulink block above and the proportional  $K_p$ , integral  $K_i$ , and derivative  $K_d$  coefficients is enter in the PID controller block as illustrated in Figure 4.4.

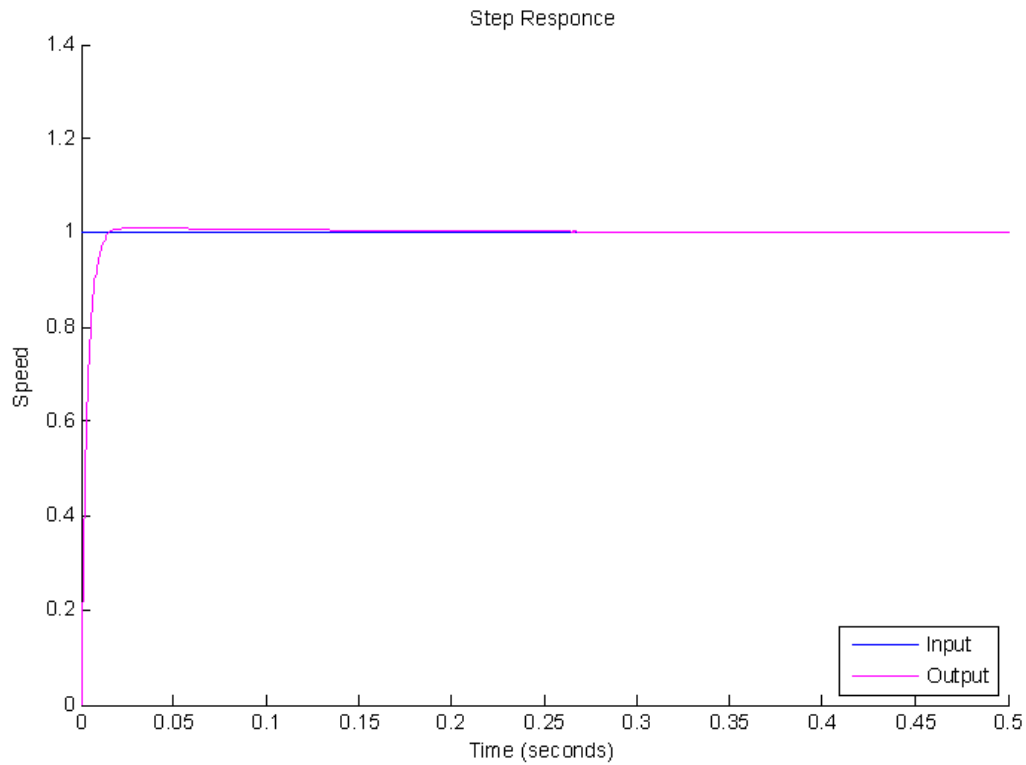


**Figure 4.4** Detail Simulink Block of the System with PID Controller

Assigning the reference point of the DC motor speed at  $r(t) = 1$  rad, the modeling was performed and Figure 4.5 illustrated the dynamic output of the system.

The commonly performance criteria to be attained are the stability with the desired stability margin in the full operating envelope, the robustness to parameter variation and changes, tracking and disturbance attenuation, dynamic and steady state accuracy, and dynamic performance specification imposed on the states and the transient response. For the DC motor system studied and designed, the settling time  $T_s$  is 29.55ms, the maximum overshoot %OS is 1.2%, the rise time  $T_r$  is 7.3828ms and the delay time  $T_d$  is 2.453ms.

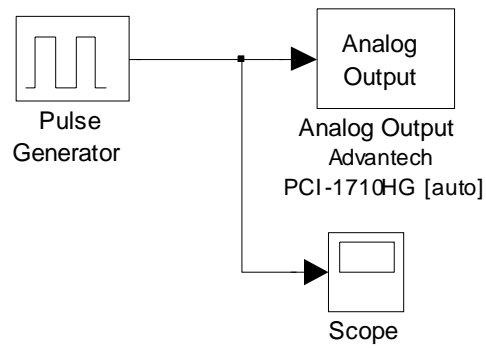




**Figure 4.5** Output of DC Motor without PID Controller

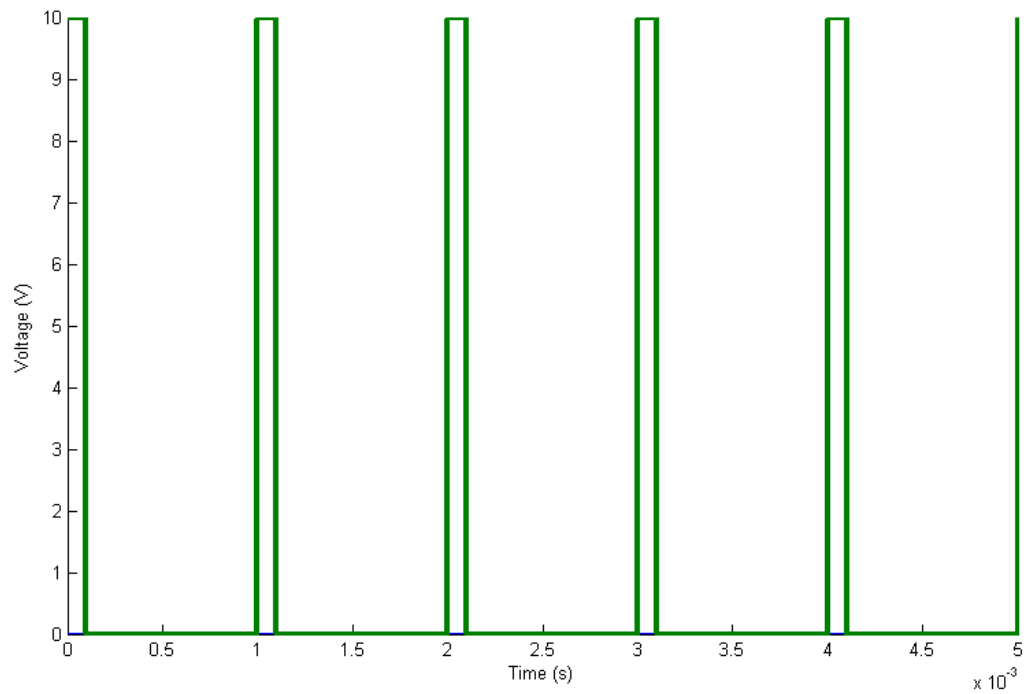
#### 4.4 Experiment without PID controller

After simulating the system with without PID controller, the actual DC motor is applied to see the weather the simulation coefficient of the proportional  $K_p$ , integral  $K_i$ , and derivative  $K_d$  will performed or needed more tuning. In this section, the DC motor will be connected to the computer through a driver and the DAQ card. The DAQ card will send the information or signal to the driver and the DC motor will turn. Due to the nature of the driver where it PWM signal in order to pass the voltage to the DC motor it is developed in Figure 4.6 the Simulink block to control the speed of the motor using pulse generator where the duty cycle is manipulated.

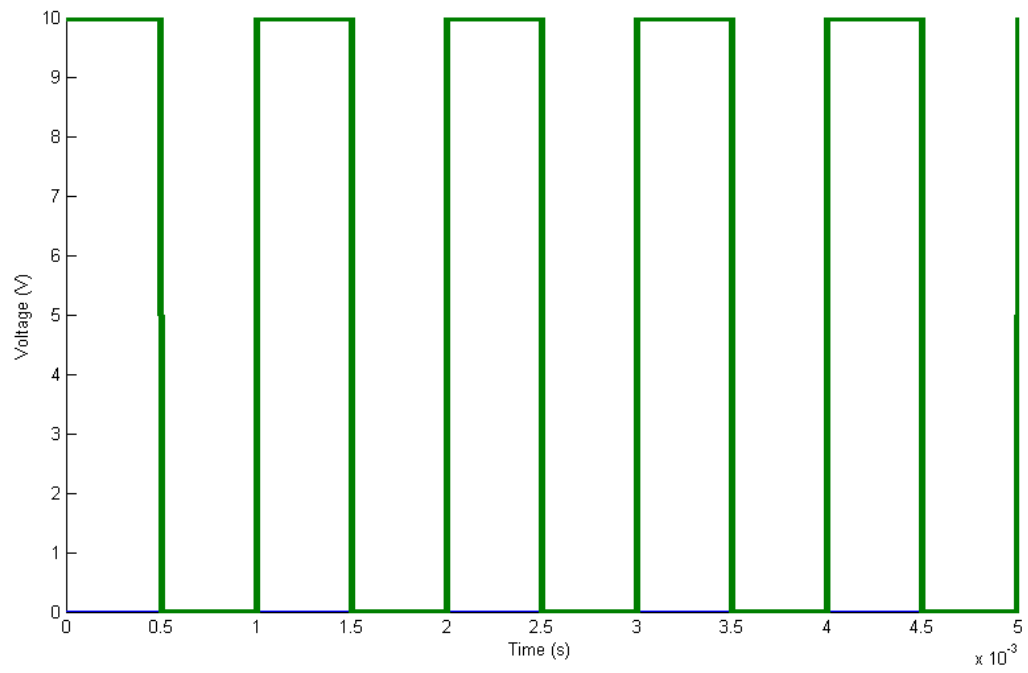


**Figure 4.6** Simulink Block of Experiment without PID

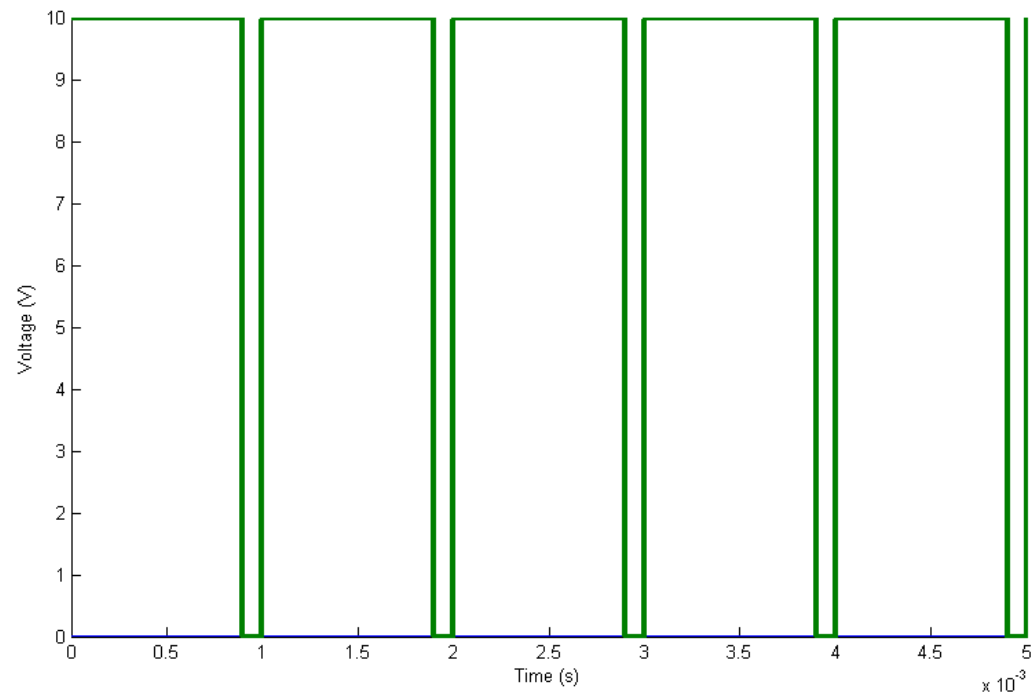
The square wave is set to frequency of 1 kHz and the voltage applied to the DC motor through the driver is 30 Volt. The pulse generated from the pulse generator is as shown in Figure 4.7 to Figure 4.9. The speed and voltage for every 10% of duty cycle is recorded in Table 4.2.



**Figure 4.7** 10% Duty Cycle Pulse



**Figure 4.8** 50% Duty Cycle Pulse

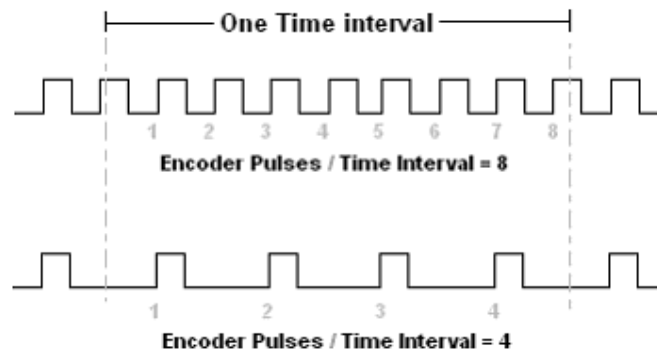


**Figure 4.9** 90% Duty Cycle Pulse

**Table 4.2:** Speed and Voltage for every 10% duty cycle

<b>Duty Cycle (%)</b>	<b>Speed (rpm)</b>	<b>Speed Increase (rpm)</b>	<b>Voltage (V)</b>	<b>Voltage Increase (V)</b>
<b>0</b>	<b>0</b>	<b>0</b>	<b>0</b>	<b>0</b>
10	240	240	3.0	3.0
20	480	240	6.0	3.0
30	720	240	9.0	3.0
40	960	240	12.0	3.0
50	1200	240	15.0	3.0
60	1440	240	18.0	3.0
70	1680	240	21.0	3.0
80	1920	240	24.0	3.0
90	2160	240	27.0	3.0
100	0	0	0	0

The speed of the DC motor was measured through an encoder where a square wave pulse was the output. There are multiple ways to determine the angular velocity of a quadrature Encoder. From the develop square wave pulse output from the encoder, the number of quadrature Encoder pulses in a fixed time interval is counted to estimate the velocity of the encoder, Figure 4.10 below demonstrates this procedure. This method is appropriate for high speed applications.

**Figure 4.10** Velocity Estimation

Once the number of pulses in a fixed time interval is measured the angular velocity of the quadrature Encoder can be calculated using the following formula:

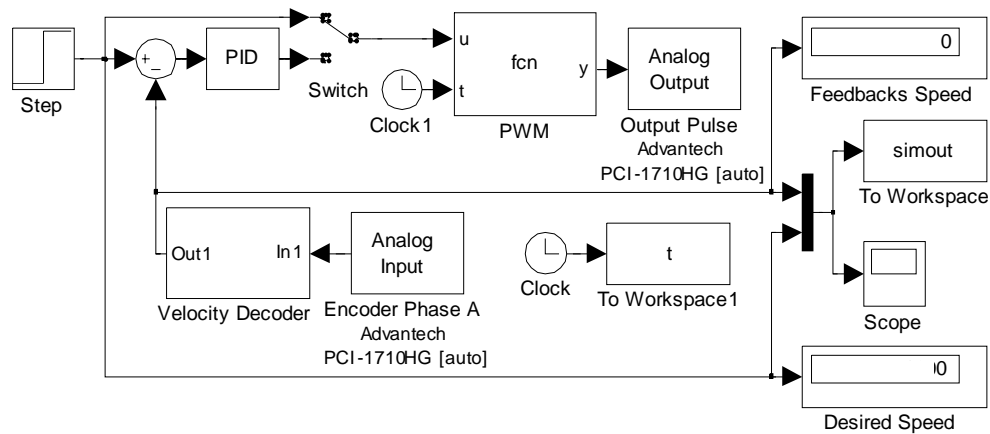
$$Velocity = \frac{\frac{Encoder Pulses}{Pulses Per Revolution} \times \frac{60 sec}{1 min}}{Fixed Time Interval (sec)} (RPM)$$

Where, “Encoder Pulses” is the number of quadrature encoder pulses received in the Fixed Time Interval and the ‘Pulse per Revolution’ is the number of pulse in 1 revolution of the encoder which is 240pulse Per Revolution base on the datasheet

From Table 4.2, the increase on speed and voltage is constant. For every 10% of duty cycle the speed increase and amount of 240 rpm and 3 Volt. The reason at duty cycle 100% the DC motor speed is 0 rpm is because the driver used only accepts pulse at the input in order to control the voltage pass to the DC motor. At 100% duty cycle the square wave pulse become a constant value of 10V therefore the drive does not operate. From the table we can figure out the maximum speed of the DC motor at 30 Volt as if it was connected directly to the 30 Volt voltage supplies which is 2400 rpm.

#### 4.5 Experiments with PID Controller

During the development of the Simulink block for the experiment PID controller with the DC motor, there were few problem encounter and solve but the end result were a problem that was unsolved. Figure 4.11 is the generated Simulink Block to be run with the real DC motor.



**Figure 4.11** Complete Simulink Block of the Experiment

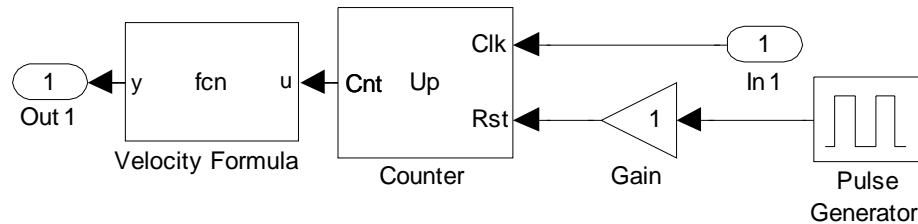
Due to the nature of the driver, the Simulink developed needed to create a PWM where the duty cycle is controlled but the PID base on the input speed. To get duty cycle a MATLAB `SQUARE(T)` is used. `SQUARE(T)` generates a square wave with period  $2\pi$  for the elements of time vector  $T$ . `SQUARE(T)` is like `SIN(T)`, only it creates a square wave with peaks of +1 to -1 instead of a sine wave. `SQUARE(T,DUTY)` generates a square wave with specified duty cycle. The duty cycle, `DUTY`, is the percent of the period in which the signal is positive. In order to create square wave of frequency of 1 kHz a function as such is create `pwm = square(2*pi*1000*t,u/2400*100)`, where `u` is the desired speed. Then this function inserted into the Embedded MATLAB Function.

In order to compare the desired speed with the actual speed of the DC motor, so that the PID controller can to calculate the error. The velocity measurement as been discussed in previous section, a velocity decoder is created. The velocity decoder is as show in Figure 4.12. The square wave pulse generated from the encoder is sent to the DAQ card trough the real-time window target Analog Input and to the velocity decoder In1. The signal is send to the counter and the positive pulse is count for every 1ms where another pulse reset the count for every 1.1ms so that the count doest continue till infinity. The counted value of the encoder pulse then is send to the Velocity Formula where it only collect the data from the counter every 1ms only so that the count value us calculated for every count value. Using the

previous discussed formula to calculate the motor speed is used in this Embedded MATLAB Function of Velocity Formula. The function is as follow:

$$y = ((u/250)*60)/0.001;$$

Where  $u$  is the total counted value of the positive pulse or rising edge of the encoder square wave signal.  $y$  is then compare with the desired speed of the DC motor and send to the PID controller to evaluate and fix the DC motor speed if there is an error.



**Figure 4.12** Velocity Decoder Subsystem Simulink Block

It was stated before that the experiment Simulink block had an unsolved problem where it was

Function output 'y' cannot be of MATLAB type.

Function 'Embedded MATLAB Function' (#30.0.176), line 1, column 1:

"function y = fcn(u,t)"

Errors occurred during parsing of Embedded MATLAB function 'Embedded MATLAB Function'(#30)

```
Embedded MATLAB Interface Error: Errors occurred during  
parsing of Embedded MATLAB function 'Embedded  
MATLAB Function'(#30).
```

```
Embedded MATLAB Interface Error:
```

In order to solve the problem it is needed to solved this problem or find other way to created the PWM in Simulink or use a driver with build in PWM generator.



## **CHAPTER 5**

### **CONCLUSION AND RECOMMENDATION**

#### **5.1 Conclusion**

MATLAB and Simulink is very user-friendly software, through which control system is design using various block provided. Simulink save a lot of time by avoiding hundred lines of coding. MATLAB and Simulink are used for simulation and for designing real-time model. There is an inherent advantage in using Simulink to model the control system. It saves time and effort, allowing the engineer to design the system in a straightforward manner, rather than wasting time writing source code from scratch. Only recently has Simulink had the capability to directly target hardware. It was now possible to create Simulink models for motor testing, open-loop system design, as well as closed-loop system design without writing any lines of code.

The basic aim of this thesis is to control the speed of a DC motor using PID controllers and is accomplished with desired specifications. The block diagram of a DC motor was developed and by using Simulink, a toolbox extension of the MATLAB program, the block diagram was simulated with expected waveforms output. The simulation and modeling of the DC motor also gave an inside look of the

expected output when testing the actual DC motor. The results from the simulation were never likely to occur in real-life condition due to the response times and condition of the actual motor. A PID controller were design suing Ziegler Nichols methods and trial and error method Then these controllers are used for controlling the speed of a DC Motor, during simulation and real-time closed loop operation. During simulation, controllers provided output with following specifications:

- Steady-state error of 0.001
- Rise time of 7.3828ms
- Delay time of 2.453ms
- % overshoot of 1.2%

Due to the problem encountered when conducting the experiment of the actual DC motor there was no data for the controlled DC motor accept for the speed and voltage of the uncontrolled DC motor. The uncontrolled DC motor show that for every 10% of the duty cycle there is an increase of 240 rpm of the DC motor speed and 3 Volt of DC motor voltage In order to control the DC motor, the width of the duty cycle must be controlled to get the desired speed.

## **5.1 Future Recommendation**

Lots of future work can be done to exploit the advantages of MATLAB and Simulink and their hardware targeting capabilities. This thesis used a simple structure of PID controller; a complicated structure can be chosen to obtain better output. There are various advances tuning method such as Robust Adaptive PID (RaPID) could be used in designing PID controller other than Ziegler Nichols methods and trial and error methods. An intelligent controller could also be used for speed control of DC drives, so the combination of intelligent controller and PID controller can be used for better control of speed.

For future studies for student to test and design a controller, a closed – loop control of DC motor via internet could be developed. This way student or event lecture could test their controller without event needing to go to the laboratory to collect data or set up the equipment, they just needed to communicate their controller with the server and the program and the data is updated to a web page for easy access and inconvenient.

## REFERENCE

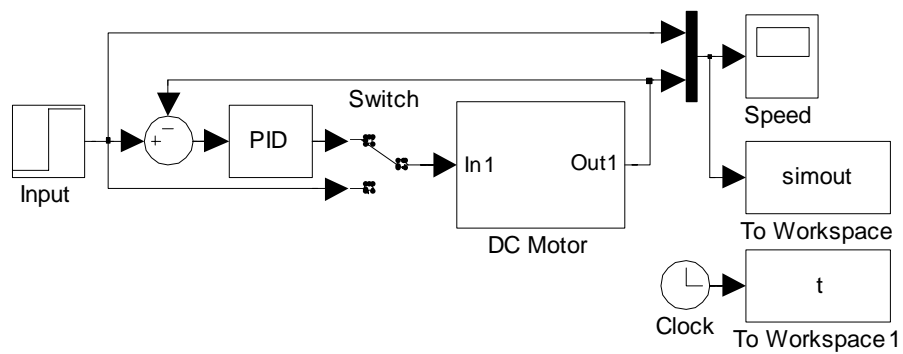
- [1] 9 January 2008, citing internet source URL  
[http://en.wikipedia.org/wiki/PID\\_controller](http://en.wikipedia.org/wiki/PID_controller)
- [2] Lyshevski, S. E., *Electromechanical Systems, Electric Machines, and Applied Mechatronics*: CRC press. 1999
- [3] 10 January 2008, citing internet source URL  
[http://en.wikipedia.org/wiki/Electric\\_motor#DC\\_motors](http://en.wikipedia.org/wiki/Electric_motor#DC_motors)
- [4] 10 January 2008, citing internet source URL  
[http://en.wikipedia.org/wiki/Brushless\\_DC\\_electric\\_motor](http://en.wikipedia.org/wiki/Brushless_DC_electric_motor)
- [5] 9 January 2008, citing internet source URL  
[http://www.actc-control.com/resource\\_centre/tech\\_pid.html#Introduction](http://www.actc-control.com/resource_centre/tech_pid.html#Introduction)
- [6] 15 January 2008, citing internet source URL  
[http://www.measurementcomputing.com/daq\\_card.htm](http://www.measurementcomputing.com/daq_card.htm)
- [7] 15 September 2008, citing internet source URL  
[http://en.wikipedia.org/wiki/Control\\_theory](http://en.wikipedia.org/wiki/Control_theory)
- [8] Gandra, D., *Real-Time of Robust PID Controller For Speed Control of DC Motor*, Master thesis, Texan A&M University-Kingsville; 2006

- [9] Jingwei Xu, *Fuzzy PID Control through Optimization: A New Method for PID Control*, Doctor Thesis, Marquette University, Milwaukee, Wisconsin; 2006
- [10] Min Xu, *High Performance and Robust Control*, Doctor Thesis, Florida Atlantic University, Boca Raton, Florida; 1996
- [11] Kamalasdan, S., and Hande, A., *A PID Controller for Real-Time DC Motor Speed Control using the C505C Microcontroller*, pp. 34-39
- [12] Chau, K.T. and Chan, C.C., *A Spice Compatible Model of Permanent Magnet DC Motor Drives*, IEEE Catalogue No. 95TH8025, Hong Kong, pp. 477-482; 1995
- [13] Seller, D., *an Overview of Proportional Plus Integral Plus Derivative Control and Suggestions for Its Successful Application and Implementation*, Portland Energy Conservation Inc, Portland, Oregon
- [14] Esposito, Joel M., Feemster, M. G., and Watkins, J. M., *Role of a MATLAB Real-Time Hardware Interface Within a Systems Modeling Course, Proceedings of the 2004 American Society for Engineering Education Annual Conference & Exposition*, United States Naval Academy; 2003
- [15] Samaranayake, L., Alahakoon, S., *Closed – loop Speed Control of a Brushless DC Motor via Ethernet*
- [16] Babuška, R., and Stramigioli, S., *Matlab and Simulink for Modeling and Control*, Delft University of Technology; 1999
- [17] Kamalasadan, S., *Real Time Speed Control of a DC Motor using C505CMicrocontroller*, The University of Toledo; 2001

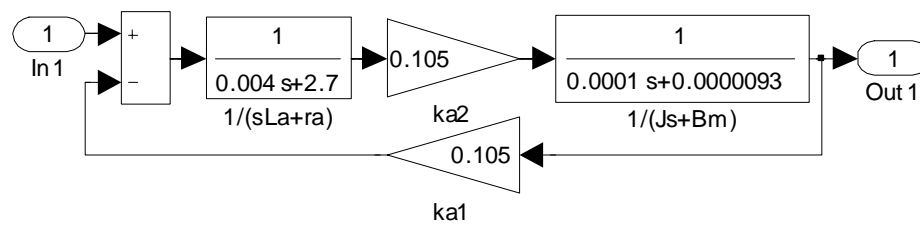
- [18] Klee, A., *Development of a Motor Speed Control System Using Matlab and Simulink, Implemented with a Digital Signal Processor*, Bachelor Thesis, University of Central Florida; 2005
  
- [19] Howe, T., K., *Evaluation of the transient response of a DC motor using MATLAB/SIMULINK*, Bachelor Thesis, University of Queensland; 2003
  
- [20] Aung, W., P., *Analysis on Modeling and Simulink of DC Motor and its Driving System Used for Wheeled Mobile Robot*, *International Journal of Computer, Information, and Systems Science, and Engineering* Volume 2 Number 1, pp. 22-29

## APPENDIX A

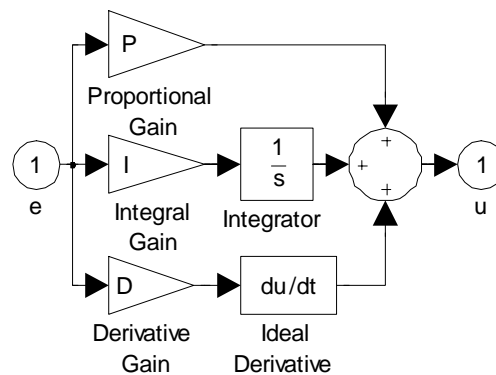
Simulink Block of PID Control DC Motor (Simulation)



Simulink Block of DC Motor



Simulink Block of PID Controller







## APPENDIX C

### Embedded MATLAB Function

```
function pwm = fcn(u,t)
% This block supports the Embedded MATLAB subset.
% See the help menu for details.

pwm = square(2*pi*1000*t,u/2400*100);
```

```
function y = fcn(u)
% This block supports an embeddable subset of the MATLAB
language.
% See the help menu for details.

y = ((u/250)*60)/0.001;
```

### MATLAB Command

```
plot(t,simout)
plot(t,scope)
```