

DEVELOPING LQR TO CONTROL SPEED MOTOR USING MATLAB GUI

SHAMSUL NIZAM BIN MOHD YUSOF @ HAMID

UNIVERSITI MALAYSIA PAHANG

UNIVERSITI MALAYSIA PAHANG

**BORANG PENGESAHAN STATUS TESIS<sup>♦</sup>**

**JUDUL:** **DEVELOPING LQR TO CONTROL SPEED MOTOR  
USING MATLAB GUI**

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

Saya **SHAMSUL NIZAM BIN MOHD YUSOF @ HAMID (850428-03-6305)**  
(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:

**LOT 355 KG. MENTERA BANGGU  
16150 KOTA BHARU  
KELANTAN**

**EN. NIK MOHD KAMIL NIK YUSOFF**  
( Nama Penyelia )

Tarikh: **12 NOVEMBER 2008**

Tarikh: : **12 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 (Electronics)”

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

Name : HASZURAI DAH BINTI ISHAK

Date : 12 NOVEMBER 2008

# **DEVELOPING LQR TO CONTROL SPEED MOTOR USING MATLAB GUI**

**SHAMSUL NIZAM BIN MOHD YUSOF @ HAMID**

**A report submitted in partial fulfillment of the  
requirements for the award of the degree of  
Bachelor of Electrical (Electronics) Engineering**

**Faculty of Electrical and 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 : SHAMSUL NIZAM BIN MOHD YUSOF @ HAMID

Date : 10 NOVEMBER 2008

*To my beloved parents and my siblings, I'm nothing without them.*

## **ACKNOWLEDGEMENT**

During doing this thesis, I found myself around with a lot of great people. Those people have helped me a lot in doing this research directly or in directly. Their contribution has helped me in understanding about my project thoroughly.

I would like to say thank to my supervisor, Madam Haszuraidah Binti Ishak for her support, guide, advices and determination in guiding me to finish my final year project and this thesis to. I would also like to express my gratitude to my colleagues, for their intention on helping me in any sort of way.

I, myself are fully in debt with Faculty of Electrical and Electronics (FKEE), for providing me necessary components, information and funding my project. Without their helped, this project was deeming to be unfinished.

Last but not least, I would like to say my gratitude toward to my family member for their support and encouragement. I am grateful to have them all.

## **ABSTRACT**

This project is mainly upon developing Linear Quadratic Regulator (LQR) controller and controlling it through software. In the software part, MATLAB GUI is been implemented to control the whole LQR system. In addition, a servo motor is attached as a hardware module to show the resultant of the LQR controller. The ability of the controller on the servo motor is, it is capable to manipulate and control the rotating speed of the motor. Thus it could also regulate the value from the error that occurs at output so that the value is stabilized as same as the input. Application of feedback system is applied in the MATLAB GUI itself before interfacing it with the servo motor using DAQ Card. The output from the MATLAB is then sent to the input of the DAQ Card through the Analog pin. A generation of signal from output at DAQ Card through the Analog pin subsequently enters to the motor driver. Driving the motor with the signal provided, it will send a feedback signal back to the system to be compared with the initial input. The whole system built from the simulink modeling through the software MATLAB.

## **ABSTRAK**

Projek ini boleh dikategorikan ke dalam dua bahagian utama, pengaturcaraan dan system pengawal servo motor. Untuk system pengawal, “Linear Quadratic Regulator” dan untuk programnya pula, Matlab GUI digunakan. System pengawal ini akan diaplikasi ke dalam program Matlab GUI bagi membolehkan pengguna mengawal dan memanipulasi kelajuan atau putaran servo motor tersebut. System pengawal yang telah diaplikasi ke dalam program akan disambungkan ke servo motor menggunakan “serial and parallel port”. Signal keluaran dari Matlab akan dihantar ke “DAQ Card” melalui pin Analog. Kemudian “feedback” dari servo motor akan di hantar ke sistem pengawal untuk dibandingkan dengan masukan dan seterusnya akan diubah sehingga “feedback” tadi akan sama nilainya dengan masukan.

# TABLE OF CONTENTS

<b>CHAPTER</b>	<b>TITLE</b>	<b>PAGE</b>
	<b>DECLARATION</b>	ii
	<b>DEDICATION</b>	iii
	<b>ACKNOWLEDGEMENT</b>	iv
	<b>ABSTRACT</b>	v
	<b>ABSTRAK</b>	vi
	<b>TABLE OF CONTENTS</b>	vii
	<b>LIST OF TABLES</b>	xi
	<b>LIST OF FIGURES</b>	xii
	<b>LIST OF ABBREVIATIONS</b>	xiv
	<b>LIST OF APPENDICES</b>	xv
<b>1</b>	<b>INTRODUCTION</b>	
1.1	General	1
1.2	Problem Statement	2
1.2.1	Current Controller and Software	2
1.2.2	Problem Solving	3
1.3	Objectives	3
1.4	Scope of Project	4
<b>2</b>	<b>LITERATURE REVIEW</b>	
2.1	Linear Quadratic Regulator	5
2.2	Data Acquisition	7
2.3	Servo Motor	10

### 3 METHODOLOGY

3.1	SOFTWARE	12
3.1.1	Real Time Target Setup	13
3.1.2	Installation and Configuration	15
3.1.2.1	C Compiler	15
3.1.2.2	Installation the Kernel	17
3.1.2.3	Testing the Installation	18
3.1.3	Procedures of Creating Real Time Applications	19
3.1.3.1	Creating a Simulink Model	19
3.1.3.2	Entering Configuration Parameters for Simulink	26
3.1.3.3	Entering Simulation Parameters for Real-Time Workshop	28
3.1.3.4	Creating a Real-Time Application	30
3.1.3.5	Running a Real-Time Application	31
3.1.4	Creating Graphical User Interfaces	32
3.1.4.1	GUI Development Environment	33
3.1.4.2	Starting Guide	33
3.1.4.3	The Layout Editor	34
3.1.4.4	Programming a GUI	35
3.1.5	Simulation of the servomotor	36
3.1.6	Graphical User Interface	38
3.2	HARDWARE	42
3.2.1	CLIFTON PRECISION SERVO MOTOR MODEL JDH-2250-HF-2C-E	42
3.2.2	G340 installation	43
3.2.3	G340 installation	44
3.2.3.1	Encoder hook up	44
3.2.3.2	Power supply hook up	45
3.2.3.3	Testing the encoder	45
3.2.3.4	Control input hook up	46

3.2.3.5	Testing the control inputs	46
3.2.3.6	Motor hook up	46
3.2.4	Advantech PCI-1710HG	47
3.2.5	Common Specifications	47
3.2.6	Pin Assignments	48
3.3	Linear Quadratic Regulator Development	48
3.3.1	The General State-Space Representation	48
3.3.2	Evaluating the State Space equation	51
3.4	Applying Linear Quadratic Regulator	54
<b>4</b>	<b>RESULT AND DISCUSSION</b>	
4.1	Simulation result	56
4.1.1	Simulation on the DC motor	56
4.1.2	Simulation on the Clifton Precision servo motor	58
4.2	Data Analysis	60
4.3	Tuning the value of Q	62
<b>5</b>	<b>CONCLUSION AND RECOMMENDATION</b>	
5.1	Introduction	65
5.2	Assessment of design	65
5.3	Strength and Weakness	66
5.4	Suggestion for Future Work	67
5.5	Costing & Commercialization	67
	<b>REFERENCES</b>	68
	<b>APPENDIX A</b>	69
	<b>APPENDIX B</b>	77

**LIST OF TABLE**

<b>TABLE NO.</b>	<b>TITLE</b>	<b>PAGE</b>
3.1	Comparison of Matlab GUI with other software and it's problem	13
3.2	Parameters of servomotor elements	53
4.1	Data analysis simulink of the system	61
4.2	Simulation results for three different Q	64

## LIST OF FIGURES

FIGURE NO. PAGE	TITLE	
2.1	LQR block diagram	5
2.2	Typical dc servo motor system with either encoder or resolver feedback	11
3.1	Real Time Windows Target	14
3.2	Simulink Model rtvdp.mdl	18
3.3	Create a new model	20
3.4	Empty Simulink model	20
3.5	Block Parameters of Signal Generator	21
3.6	Block Parameters of Analog Output	23
3.7	Scope Parameters Dialog Box	24
3.8	Scope Properties: axis 1	25
3.9	Completed Simulink Block Diagram	26
3.10	Configuration Parameters – Solver	27
3.11	Configuration Parameters – Hardware Implementation	28
3.12	System Target File Browser	29
3.13	Configuration Parameters – Real-Time Workshop	29
3.14	Connect to target from the Simulation menu	31
3.15	GUIDE Quick start	34
3.16	Layout Editor	35
3.17	M-File Editor	36
3.18	Block diagram for the simulation servo motor	37
3.19	Subsystem for the block diagram above	37
3.20	M-file for the system	38
3.21	Main panel	38
3.22	Supervisor credit	39
3.23	Student credit	39
3.24	Abstract of the project	40
3.25	Manual way to find the K gain	40

3.26	Simulink model	41
3.27	Find K if different motor constant	41
3.28	Servo Motor	42
3.29	Gecko drive (G340)	43
3.30	G340 block diagram	43
3.31	Advantech PCI-1710HG	47
3.32	Pin Assignment for PCI-1710HG	48
3.33	Graphic representation of state space and state vector	50
3.34	DC motor model system	51
4.1	Result for DC motor	57
4.2	Graph result for $K = [0.245006 \ 0.0801054]$	57
4.3	Result for Clifton Precision servo motor	58
4.4	Graph result for $K = [1.28138 \ 0.90017]$	59
4.5	Settling time for simulation DC motor	60
4.6	Rise time for simulation DC motor	61
4.7	Result for $Q = 1$	62
4.8	Result for $Q = 10$	63
4.9	Result for $Q = 100$	63

## LIST OF ABBREVIATIONS

GUI	-	Graphical User Interface
IEEE	-	Institute of Electrical and Electronics Engineers
LQR		Linear Quadratic Regulator
LQG		Linear Quadratic Gaussian

**LIST OF APPENDICES**

<b>APPENDIX</b>	<b>TITLE</b>	<b>PAGE</b>
A	G340 Installation	69
B	Coding Program	77

# **CHAPTER 1**

## **INTRODUCTION**

### **1.1 GENERAL**

This project is based on the controller and the software used to interface the CLIFTON PRECISION motor. By developing Linear Quadratic Regulator (LQR) using mathematical equation to get the feedback controller to control the speed of the servo motor with using Matlab GUI from Mathworks.

Matlab GUI is the one of the software that is using graphical method. Between the servo motor and Matlab GUI, DAQ Card used to interface the both of them. From the software, the data transfer first to the DAQ Card. Then the DAQ Card will convert the data to an electrical signal that acquired by the servo motor.

The drive that drives the servo motor is G340 Servodrive(x10 Step Multiplier) by Geckodrive that comes in package with the servo motor.

The servo motor is type JDH-2250-HF-2C-E CLIFTON PRECISION that has the supply voltage -0.5 to 7V interval. The output voltage of the servo motor is -0.5 to Vcc. Choosing this servo motor is because High output power relative to motor size and weight. It also suitable for the controller that has feedback, not like stepper motor which is operated open loop.

## 1.2 PROBLEM STATEMENT

As research had been done, there are lot of controller that are used. But there are some features not same as LQR features in others controller.

### 1.2.1 Current controller and software

There are a lot of controllers which can be used to control the speed of the motor such as Proportional Integral Derivatives (PID) and Fuzzy Logics. For the software, many companies has developed various software related to engineering in this day like Matlab, Visual Basic and Labview. However, the problems are:-

i. PID controller

The controller like PID need has percentage of overshoot and take some time to it's stabilizing the system. It also has the time settling that are can reach more than 1 sec. this will affect the effectiveness of the system.

ii. Software

Software like Labview has problem in term of control system because we need to download separately from the manufacturer's website the control toolkit. Then after that we could use it for the control system. The toolkit is like the add-ons to the Labview software. For the Visual Basic, we cannot do the simulation to know the result before we do it in the real-time.

### 1.2.2 Problem solving

- i. Linear quadratic regulator is the most effective controller because it regulates the error to zero and it doesn't have percentage of overshoot and time settling. So it can stabilize the system quicker than PID.
- ii. Matlab GUI is choose because it is friendly user and don't use complex coding to run it. Just drag and drop the function to use it. It also can simulate the linear system before use in real-time using Simulink. In Simulink, we need to create the system using block diagram in Matlab.

## 1.3 Objectives

First objective that has been setup for this project is in developing the controller that used. Linear Quadratic Regulator need to be develops first before can proceed to another objective or to the next step for this project. Before can do the developing, need to study first about this controller because this the first time I want to use it.

To develop LQR, there is more than one method that can be use. The purpose is to find the gain value,  $K$ . The method is like pole placement, and Algebraic Riccati equation (ARE). For this project, ARE method used because only need to find the gain value,  $K$  that is need to use to regulate the error that comes from the feedback in the linear system.

The second objective is to control the speed of the servomotor. We control the speed of the servomotor with the value of the gain,  $K$  that we obtain before. The gain will be multiplied with the output value to make the error zero and same as the input. To control the speed of the servomotor, we use Matlab GUI as the user control

panel that from there, we can control the speed by change the input as we can say the speed of the servomotor by just insert the value at the Matlab GUI.

The third one is interfacing the servomotor with the software. To doing that, a converter that converts data from computer to electrical signal called DAQ Card (Data Acquisition Card) need to be use.

## **1.4 Scope**

The most important step in this project is obtaining the state space of the motor that will use for this project first. It is because the gain value,  $K$  via Algebraic Riccati Equation (ARE) can be obtained from state space equation.

After the value of the gain  $K$  been obtained, the linear system can be created and will be simulated. The result is obtained from the simulation of the system, and the value of the gain can regulate the error or not can be determined. If not, find another value of  $K$  by change the value of the disturbance,  $Q$  in the ARE equation.

## CHAPTER 2

### LITERATURE REVIEW

#### 2.1 Linear Quadratic Regulator (LQR)

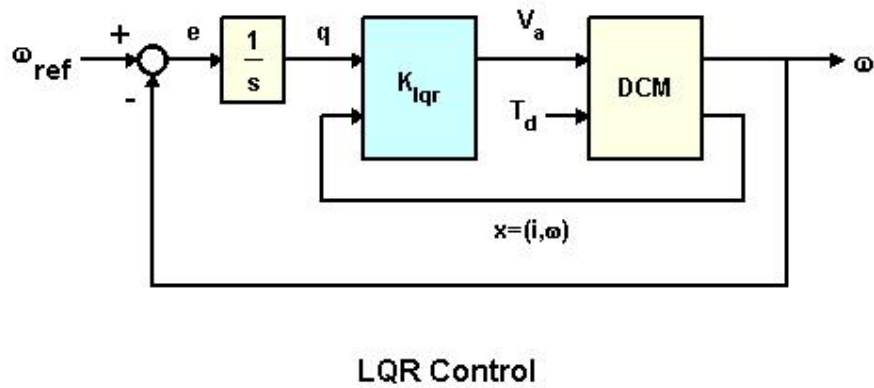


Figure 2.1: LQR block diagram

The theory of optimal control is concerned with operating a dynamic system at minimum cost. The case where the system dynamics are described by a set of linear differential equations and the cost is described by a quadratic functional is called the LQ problem. One of the main results in the theory is that the solution is provided by the linear-quadratic regulator (LQR), a feedback controller whose equations are given below. The LQR is an important part of the solution to the LQG problem. Like the LQR problem itself the LQG problem is one of the most fundamental problems in control theory.

In layman's terms this means that the settings of a (regulating) controller governing either a machine or process (like an airplane or chemical reactor) are found by using a mathematical algorithm that minimizes a cost function with weighting factors supplied by a human (engineer). The "cost" (function) is often defined as a sum of the deviations of key measurements from their desired values. In effect this algorithm therefore finds those controller settings that minimize the undesired deviations, like deviations from desired altitude or process temperature. Often the magnitude of the control action itself is included in this sum as to keep the energy expended by the control action itself limited.

In effect, the LQR algorithm takes care of the tedious work done by the control systems engineer in optimizing the controller. However, the engineer still needs to specify the weighting factors and compare the results with the specified design goals. Often this means that controller synthesis will still be an iterative process where the engineer judges the produced "optimal" controllers through simulation and then adjusts the weighting factors to get a controller more in line with the specified design goals.

The LQR algorithm is, at its core, just an automated way of finding an appropriate state-feedback controller. And as such it is not uncommon to find that control engineers prefer alternative methods like full state feedback (also known as pole placement) to find a controller over the use of the LQR algorithm. With these the engineer has a much clearer linkage between adjusted parameters and the resulting changes in controller behavior. Difficulty in finding the right weighting factors limits the application of the LQR based controller synthesis. [6]

Linear Quadratic Regulator (LQR) is the most common approach to modern control design, and one of the controller that are commonly used by users to control the system besides Proportional Integral Derivatives (PID) because its stability. For a LTI system:

$$\begin{aligned} \dot{x} &= Ax + Bu \\ y &= Cx + Du \end{aligned} \quad (2.1)$$

The technique involves choosing a control law  $u = -Kx$  which stabilizes the origin (i.e., regulates  $x$  to zero). The gain  $K$  which solves the LQR problem is

$$K = R^{-1} B^T P^* \quad (2.2)$$

where  $P^*$  is the unique, positive semidefinite solution to the algebraic Riccati equation (ARE):

$$A^T P + PA - PBR^{-1}B^T P + Q = 0 \quad (2.3)$$

## 2.2 Data acquisition

Data acquisition is the sampling of the real world to generate data that can be manipulated by a computer. Sometimes abbreviated DAQ or DAS, data acquisition typically involves acquisition of signals and waveforms and processing the signals to obtain desired information. The components of data acquisition systems include appropriate sensors that convert any measurement parameter to an electrical signal, which is acquired by data acquisition hardware.

Acquired data are displayed, analyzed, and stored on a computer, either using vendor supplied software, or custom displays and control can be developed using various general purpose programming languages. LabVIEW, which offers a graphical programming environment optimized for data acquisition and MATLAB provides a programming language but also built-in graphical tools and libraries for data acquisition and analysis.

Data acquisition begins with the physical phenomenon or physical property of an object (under investigation) to be measured. This physical property or phenomenon could be the temperature or temperature change of a room, the intensity

or intensity change of a light source, the pressure inside a chamber, the force applied to an object, or many other things. An effective data acquisition system can measure all of these different properties or phenomena.

A transducer is a device that converts a physical property or phenomenon into a corresponding measurable electrical signal, such as voltage, current, change in resistance or capacitor values, etc. The ability of a data acquisition system to measure different phenomena depends on the transducers to convert the physical phenomena into signals measurable by the data acquisition hardware. Transducers are synonymous with sensors in DAQ systems. 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 (also called logic signals sometimes) 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 to 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.

Analog signals tolerate almost no cross talk and so are converted to digital data, before coming close to a PC or before traveling along long cables. For analog data to have a high signal to noise ratio, the signal needs to be very high, and sending  $\pm 10$  Voltages along a fast signal path with a 50 Ohm termination requires powerful drivers. With a slightly mismatched or no termination at all, the voltage along the cable rings multiple time until it is settled in the needed precision. Digital data can have  $\pm 0.5$  Volt. The same is true for DACs. Also digital data can be sent over glass fiber for high voltage isolation or by means of Manchester encoding or similar

through RF-couplers, which prevent net hum. Also as of 2007 16bit ADCs cost only 20 \$ or €.

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. Usually the space on the back of a PCI card is too small for all the connections needed, so an external breakout box is required. The cable between this Box and the PC is expensive due to the many wires and the required shielding and because it is exotic. 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. For example: Waiting for a trigger, starting the ADC, looking up the time, waiting for the ADC to finish, move value to RAM, switch multiplexer, get TTL input, let DAC proceed with voltage ramp. As 16 bit ADCs and DACs and OpAmps and sample and holds with equal precision as of 2007 only run at 1 MHz, even low cost digital controllers like the AVR32 have about 100 clock cycles for bookkeeping in between. Reconfigurable computing may deliver high speed for digital signals. Digital signal processors spend a lot of silicon on arithmetic and allow tight control loops or filters. The fixed connection with the PC allows for comfortable compilation and debugging. Using an external housing a modular design with slots in a bus can grow with the needs of the user. High speed binary data needs special purpose hardware called Time to digital converter and high speed 8 bit ADCs are called oscilloscope Digital storage oscilloscope, which are typically not connected to DAQ hardware, but directly to the PC.

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.[4]

### 2.3 Servo Motor

A servomechanism or servo is an automatic device which uses error-sensing feedback to correct the performance of a mechanism. The term correctly applies only to systems where the feedback or error-correction signals help control mechanical position or other parameters. For example an automotive power window control is not a servomechanism, as there is no automatic feedback which controls position—the operator does this by observation. However, if the operator and the window motor could be considered together, perhaps they as an entity could be said to operate via a servomechanism. By contrast the car's cruise control uses closed loop feedback, which classifies it as a servomechanism.

Servomechanisms may or may not use a servomotor. For example a household furnace controlled by thermostat is a servomechanism, yet there is no closed-loop control of a servomotor.

A common type of servo provides position control. Servos are commonly electrical or partially electronic in nature, using an electric motor as the primary means of creating mechanical force. Other types of servos use hydraulics, pneumatics, or magnetic principles. Usually, servos operate on the principle of negative feedback, where the control input is compared to the actual position of the mechanical system as measured by some sort of transducer at the output. Any difference between the actual and wanted values or an "error signal" is amplified and used to drive the system in the direction necessary to reduce or eliminate the error. An entire science known as control theory has been developed on this type of system.

Servomechanisms were first used in military fire-control and marine navigation equipment. Today servomechanisms are used in automatic machine tools, satellite-tracking antennas, automatic navigation systems on boats and planes, and antiaircraft-gun control systems. Other examples are fly-by-wire systems in aircraft which use servos to actuate the aircraft's control surfaces, and radio-controlled models which use RC servos for the same purpose. Many autofocus cameras also use

a servomechanism to accurately move the lens, and thus adjust the focus. A modern hard disk drive has a magnetic servo system with sub-micrometer positioning accuracy.

Typical servos give a rotary (angular) output. Linear types are common as well, using a screw thread or a linear motor to give linear motion.

Another device commonly referred to as a servo is used in automobiles to amplify the steering or braking force applied by the driver. However, these devices are not true servos, but rather mechanical amplifiers.

Servo motor is one of the devices that have the applications where precise positioning and speed required. The big advantage of the servo motor is that servos are operated "closed loop". This means feedback is required from the motor, that's why this system is sensitivity to disturbances and have ability to correct these disturbances. [7]

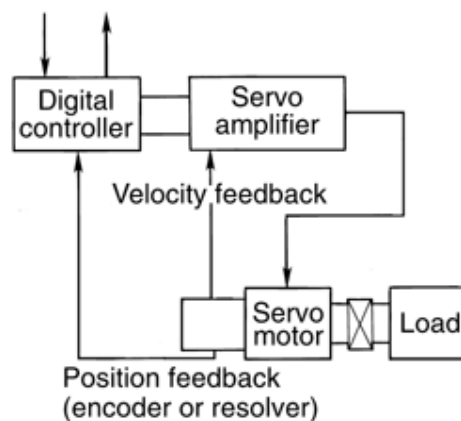


Figure 2.2: Typical dc servo motor system with either encoder or resolver feedback.

## CHAPTER 3

### METHODOLOGY

#### 3.1 SOFTWARE

The MATLAB software is use for this project because it allows one to perform numerical calculations and visualize the result without need for complicated and time consuming programming. This software provides an easy way to go directly from collecting data to deriving informative result. It also accurately solves the problem, to produce graphics easily and create the code efficiently.

MATLAB software is compatible with the Advantech PCI-1710HG that will work together in this project. It also supports the entire data acquisition and analysis process, including interfacing with data acquisition devices and instruments, analyzing and visualizing the data and producing presentation quality output.

The table shows the different between Matlab GUI and others software that have the same function and the problem with the GUI. The comparison between Matlab GUI and others software like Visual Basic, Labview, and C++ can be the reason why Matlab GUI was selected for this project. In the table also state the problem with the software and can be a guidance to avoid making mistake while working with the software. [5]

Table 3.1: Comparison of Matlab GUI with others software and its problem.

Matlab GUI versus others	Problems with GUI
Similar to RAD such as C++ builder and VB	Not as flexible

Most GUI work across platforms	Cross platform appearance may not be the same
Can perform most functions as traditional GUI through tricks	Often must use tricks and unfriendly techniques
Can link platform dependent code using MEX programs	MEX code GUI eliminates cross platform operation

### 3.1.1 Real Time Target Setup

Real-Time Windows Target enables to run Simulink and Stateflow model in real time on desktop or laptop PC for rapid prototyping or hardware-in-the loop simulation of control system and digital signal processing algorithms. A real-time execution can be created and controlled entirely through Simulink. Using Real-time Workshop, C code can be generated, compiled, and started real-time execution on Windows PC while interfacing to real hardware using PC I/O board. I/O device drivers are included to support an extensive selection of I/O boards, enabling to interface to sensor, actuators, and other devices for experimentation, development, and testing real-time systems. Simulink block diagram can be edited and Real-Time Workshop can be used to create a new real-time binary file. This integrated environment would implement any designs quickly without lengthy hand coding and debugging. Figure shows the required product of Real Time Windows Target.

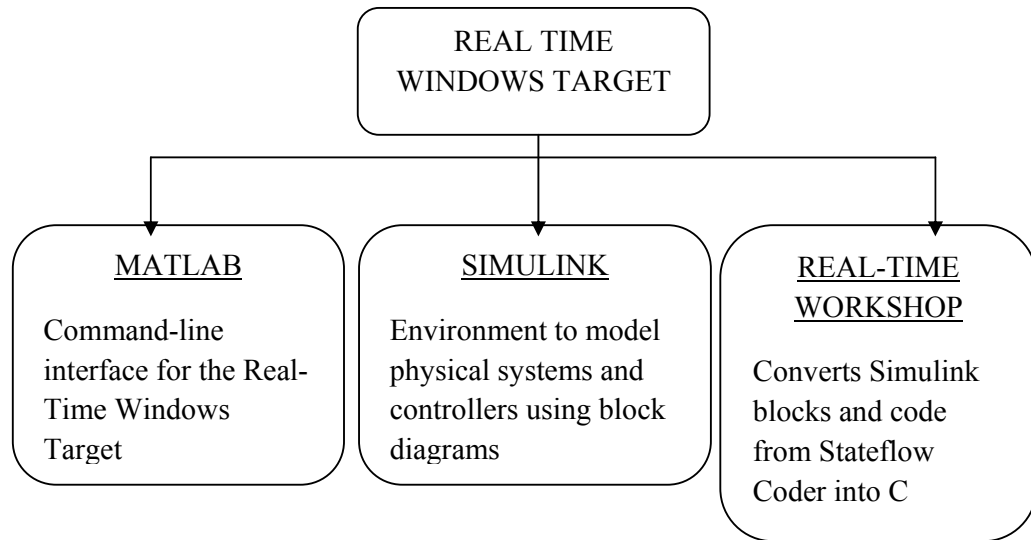


Figure 3.1: Real Time Windows Target

Real-Time Windows Target includes a set of I/O blocks that provide connections between the physical I/O board and real-time model. Hardware-in-the-loop simulations can be ran and quickly observed how Simulink model responds to real-world behavior. I/O signals can be connected using the block library for operation with numerous I/O boards.

The following types of blocks are included:

- Digital Input blocks : Connect digital input signals to Simulink block diagram to provide logical inputs.
- Digital Output blocks : Connected logical signals from Simulink block diagram to control external hardware.
- Analog Input blocks : Enable to use A/D converters that digitize analog signal for use as input to Simulink block diagram.
- Analog Output blocks : Enable Simulink block diagram to use D/A converters to output analog signal from Simulink model using I/O board(s).
- Counter Input blocks : Enable to count pulses or measure frequency using hardware counters on I/O board(s).
- Encoder Input blocks : Enable to include feedback from optical encoders.

### 3.1.2 Installation and Configuration

The Real-Time Windows Target is a self-targeting system where the host and the targeting computer are the same computer. It can be installed on a PC-compatible computer running Windows NT 4.0, Windows 2000 or Windows XP.

#### 3.1.2.1 C Compiler

The Real-Time Windows Target requires one of following C compilers which not included in with the Real Time Windows Target:

- Microsoft Visual C/C ++ compiler - - Version 5.0, 6.0 or 7.0
- Watcom C/C ++ compiler - - Version 10.6 and 11.0. During installation of Watcom C/C ++ compiler, a DOS target is specified in addition to a windows target to have necessary libraries available for linking.

After installation, the MEX utility is run to select compiler as the default compiler for building real-time applications.

Real Time Workshop uses the default C compiler to generate executable code and the MEX utility uses this compiler to create MEX-files.

This procedure is executed in order to select either a Microsoft Visual C/C ++ compiler or a Watcom C/C ++ compiler before build an application. Note, the LCC compiler is not supported:

1. `mex -setup` is typed in the MATLAB window  
 MATLAB will display the following message:  
     Please choose your compiler for building eternal  
     interface

(MEX) files. Would you like mex to locate installed compilers? ([y] / n) :

Then a letter “y” is typed.

MATLAB will display the following message:

Select a compiler:

[1]: WATCOM Compiler in c: \watcaom

[2]: Microsoft compiler in c: \visual

[0]: None

Compiler:

Next, a number is typed. For example, number 2 is typed to select the Microsoft compiler.

MATLAB will display the following message:

Please verify your choices:

Compiler: Microsoft 5.0

Location: c: \visual

Are these correct? ([y] / n)

Finally, a letter “y” is typed.

MATLAB will reset the default compiler and display the message:

The default option file:

“c:\WINNT\Profiles\username\Application

Data\MathWorks\MATLAB\mexopts.bat” is being updated.

### 3.1.2.2 Installation the Kernel

During installation, all software for the Real-Time Windows Target is copied onto hard drive. The kernel is not automatically installed. Installing the kernel sets up the kernel to start running in the background each time when the computer is started. The kernel can be installed just after the Real-Time Windows Target has been installed.

The installation of the kernel is necessary before a Real-Time Windows Target can be executed:

1. `rtwintgt -install` is typed in MATLAB window.  
 MATLAB will display the following message:  
     You are going to install the Real-Time Windows Target kernel.  
     Do you want to proceed? [y] :
  
2. The kernel installation is continued by typing a letter “y”.  
 MATLAB will install the kernel and display the following message:  
     The Real-Time Windows Target kernel has been successfully installed.  
     The computer has to be restart if a “restart” message being displayed.
  
3. The kernel should be checked whether it was correctly installed. Then, `rtwho` is typed.  
 MATLAB would display a message similar to  
     Real-Time Windows Target version 2.5.0 (C) The MathWorks, Inc.  
     1994-2003  
     MATLAB performance = 100.0%  
     Kernel timeslice period = 1ms

After the kernel being installed, it remains idle, which allows Window to control the execution of any standard Windows application. Standard Windows applications include internet browsers, word processors, MATLAB and so on. It is only during real-time execution of model that the kernel intervenes to ensure that the model is given priority to use the CPU to execute each model updating at the prescribed sample intervals. Once the model update at a particular sample interval completed, the kernel releases the CPU to run any other Windows application that might need servicing.

### **3.1.2.3 Testing the Installation**

The installation can be tested by running the model `rtvdp.mdl`. This model does not have any I/O blocks, so that this model can be run regardless of the I/O

boards in computer. Running this model would test the installation by executing Real-Time Workshop, Real-Time Windows Target and Real-Time Windows Target kernel. After the Real-Time Windows Target kernel being installed, the entire installation can be tested by building and running a real-time application. The Real-Time Windows Target includes the model `rtvdp.mdl`, which already has the correct Real-Time Workshop options selected for users:

1. `rtvdp` is typed in MATLAB window.

The Simulink model `rtvdp.mdl` window will be opened as shown in Figure 3.2

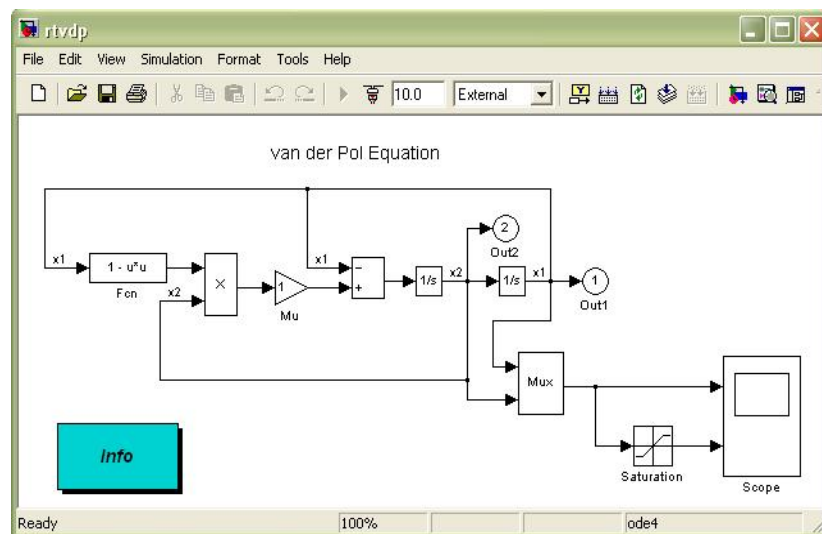


Figure 3.2: Simulink Model `rtvdp.mdl`

2. From the **Tools** menu, it should be pointed to **Real-Time Workshop**, and then clicked **Build Model**. The MATLAB window will display 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 builds procedure for
model: rtvdp

```

3. From the **simulation** menu, **External** should be clicked and followed by clicking **Connect to target**.

The MATLAB window displayed the following message:

Model rtvdp loaded

4. **Start Real-Time Code** is clicked from **Simulation** menu.  
The Scope window will display the output signals. After the Real-Time Windows Target has been successfully installed and the real-time application has been run, Scope window should indicate such a figure.
5. From **Simulation** menu, after the **Stop Real-Time Code** is clicked. The real-time application will stop running and then the Scope window will stop displaying the output signals.

### 3.1.3 Procedures of Creating Real Time Applications

#### 3.1.3.1 Creating a Simulink Model

This procedure explains how to create a simple Simulink model. This model is used as an example to learn other procedures in the Real-Time Windows Target. A Simulink model has to be created before it can run a simulation or create a real-time application:

1. Simulink is typed in the MATLAB Command Window.  
The Simulink Library Browser window is opened as shown in Figure 3.11.
2. From the toolbar, the **Create a new model** button is clicked.

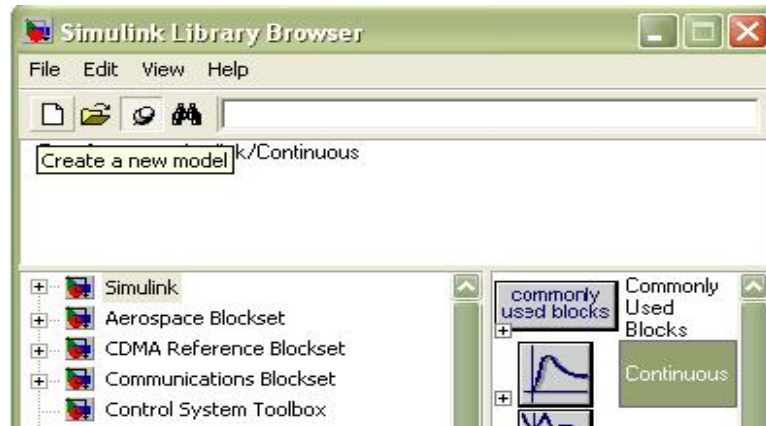


Figure 3.3: Create a new model

An empty Simulink window is opened. With the toolbar and status bar disabled, the window looks like following figure 3.12 (Figure).

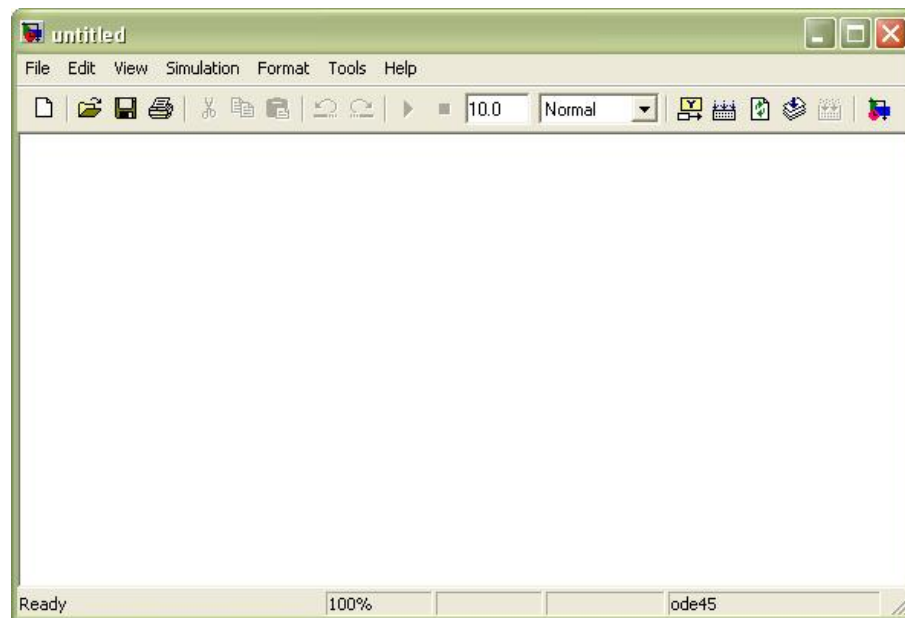


Figure 3.4: Empty Simulink model

3. In the Simulink Library Browser window, **Simulink** is double-clicked and then **Sources** is also double-clicked. Next, **Signal Generator** is clicked and dragged to Simulink window. **Sinks** is clicked. **Scope** is clicked and dragged to the Simulink window. Real-Time Windows Target is clicked. **Analog Output** is clicked and dragged 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** input is connected to the connection between **Scope** and **Signal Generator**.
5. The Signal Generator block is double clicked. The **Block Parameters** dialog box opened. From the **Wave form** list, square is selected.  
In the **Amplitude** text box, 0.25 is entered.  
In the **Frequency** text box, 2.5 are entered.  
From the **Units** list, Hertz is selected.  
The **Block Parameters** dialog box is shown in Figure 3.13.

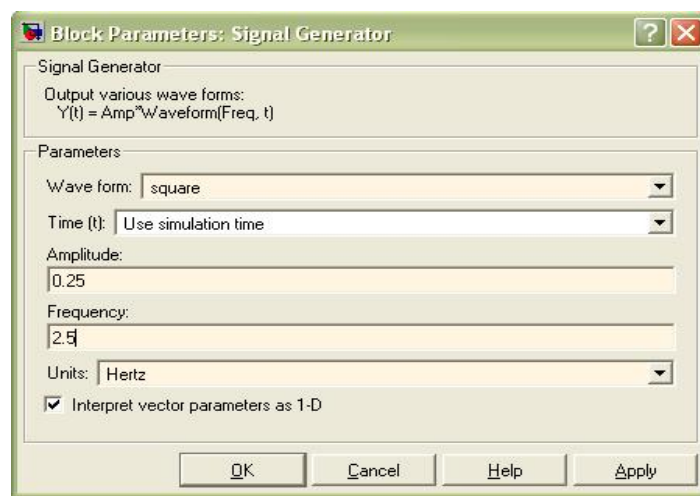


Figure 3.5: Block Parameters of Signal Generator

6. **OK** is clicked.
7. The analog output block is double clicked.  
The **Block Parameters** dialog box will open.
8. The **Install new board** button is clicked. From the list, it should be pointed to manufacturer and then clicked a board name. For example, it should be pointed to Advantech and then click PCL818.
9. One of the following is selected:
  - For an **ISA** bus board, a base address is entered. This value must match the base address switches or jumpers set on the physical board. For example, to enter a base address of 0x300 in the address box, 300 is typed. The base address also could be selected by selecting check boxes A9 through A3.

- For a **PCI** bus board, the PCI slot is entered or the Auto-detect check box is selected.
10. The **Test** button is clicked.

The Real-Time Windows Target tried to connect to the selected board and the following message would display if successful.

11. On the message box, **OK** is clicked.
12. The same value as entered in the **Fixed step size** box from the **Configuration Parameters** dialog box is entered in the Sample time box. For example, 0.001 is entered.
13. A channel vector that selected the analog input channels that are using on this board is entered in the **Output channels** box. The vector can be any valid MATLAB vector form. For example, to select analog output channel on PCL818 board 1 is entered.
14. The input range for the entire analog input channel that has been entered in the Input channels box is chosen from the **Output range** list. For example, with the PCL818 board, 0 to 5V is chosen.
15. From the Block Input signal list, the following options is chosen:
- Volts – Expected a value equal to the analog output range.
  - Normalized unipolar – Expected 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 converted from values between 0 and +1.
  - Normalized bipolar – Expected a value between -1 and +1 that is converted to the full range of output voltage regardless of the output voltage range.
  - Raw – Expected a value of 0 to  $2^n-1$ . For example, a 12-bit A/D converter would expected 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 error.

16. The initial value is entered for each analog output channel that has been entered in the **Output Channels** box. For example, if 1 is entered in the **Output Channels** box and the initial value of 0 volts is needed, 0 is entered.
17. The final value is entered for each analog channel that has been entered in **Output Channels** box. For example, if 1 is entered in the **Output Channels** box and the final value of 0 volts is needed, 0 is entered.

The dialog box would look similar to the Figure 3.14 if Volts is chosen.

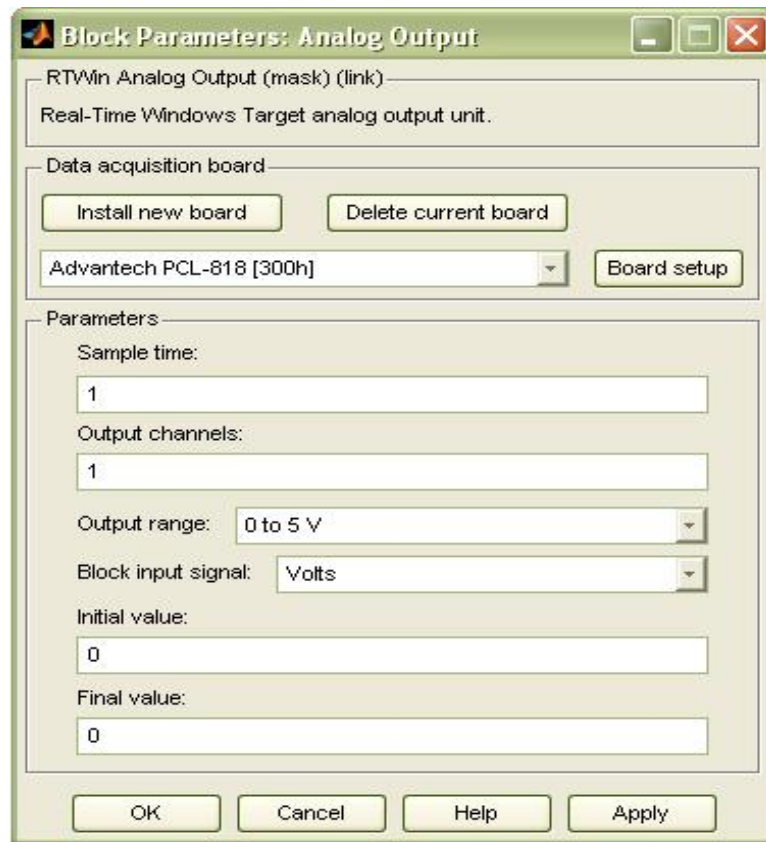


Figure 3.6: Block Parameters of Analog Output

18. One of following is executed:
  - **Apply** is clicked to apply the changes to the model and the dialog box is left open.
  - **OK** is clicked to apply the changes to the model and the Block Parameters: Analog Output dialog box will close.

19. **Parameters** dialog box is closed and the parameter values are saved with the Simulink model.
20. In the Simulink window, the Scope block is double clicked.  
A Scope window will open.
21. The Parameters button is clicked.  
A Scope parameters dialog box will open.
22. The **General** tab is clicked. The number of graphs that is needed in one Scope window is entered in the **Number of axes** box. For example, 1 is entered for a single graph. Do not select the **floating scope** check box. In the **Time range** box, upper value 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 would look like such a Figure 3.15 as shown.



Figure 3.7: Scope Parameters Dialog Box

23. One of following done:
  - **Apply** is clicked to apply the changes to the model and the dialog box is left open.
  - **OK** is clicked to apply the changes to the model and the **Scope parameters** dialog box is closed.

24. In the Scope window, it should be pointed to the y-axis and then right clicked.
25. Axes Properties is clicked from the pop-up menu.
26. The Scope properties: axis 1 dialog box is opened. 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 are entered as shown in the Figure 3.16

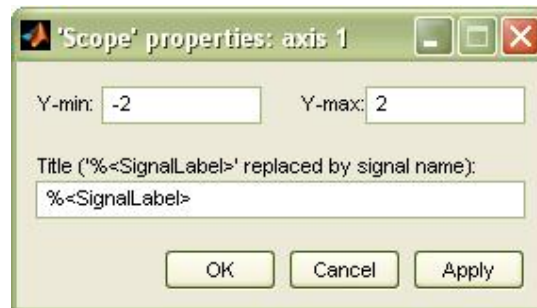


Figure 3.8: Scope Properties: axis 1

27. One of the following is done:
  - **Apply** is clicked to apply the changes to the model and the dialog box is left open.
  - **OK** is clicked to apply the changes to the model and the **Axes Parameters** dialog box is closed.

The completed Simulink block diagram is shown in Figure 3.17.

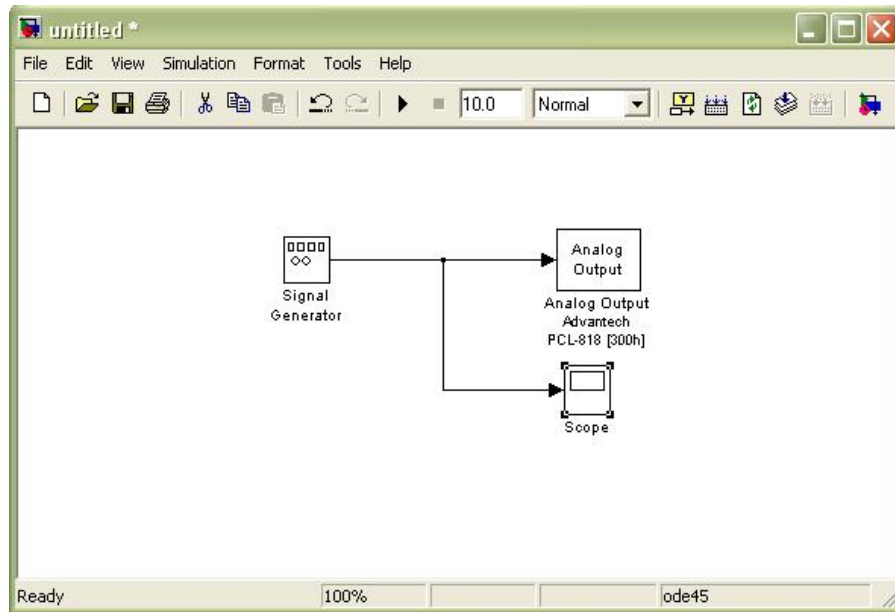


Figure 3.9: Completed Simulink Block Diagram

**Save As** is clicked from the **File** menu. The **Save As** dialog diagram box is opened. In the **File name** text box, a filename for the Simulink model is entered and **Save** is clicked. For example, `rtwin_model` is typed. Simulink saved the model in the file `rtwin_model.mdl`.

### 3.1.3.2 Entering Configuration Parameters for Simulink

The configuration parameters give information to Simulink for running a simulation. After create a Simulink model, the configuration parameters could be entered for Simulink.

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

2. In the **Start time** box, 0.0 is entered. In the **Stop time** box, the amount of time that the model needs to run is entered. For example, 99999 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** box, a sample time is entered. For example, 0.001 seconds is entered for the sample rate of 1000 samples/second.
6. From the **Tasking Mode** list, SingleTasking is chosen. Multitasking is chosen for models with blocks that have different sample times.

The **Solver** pane would look similar to the Figure 3.18.

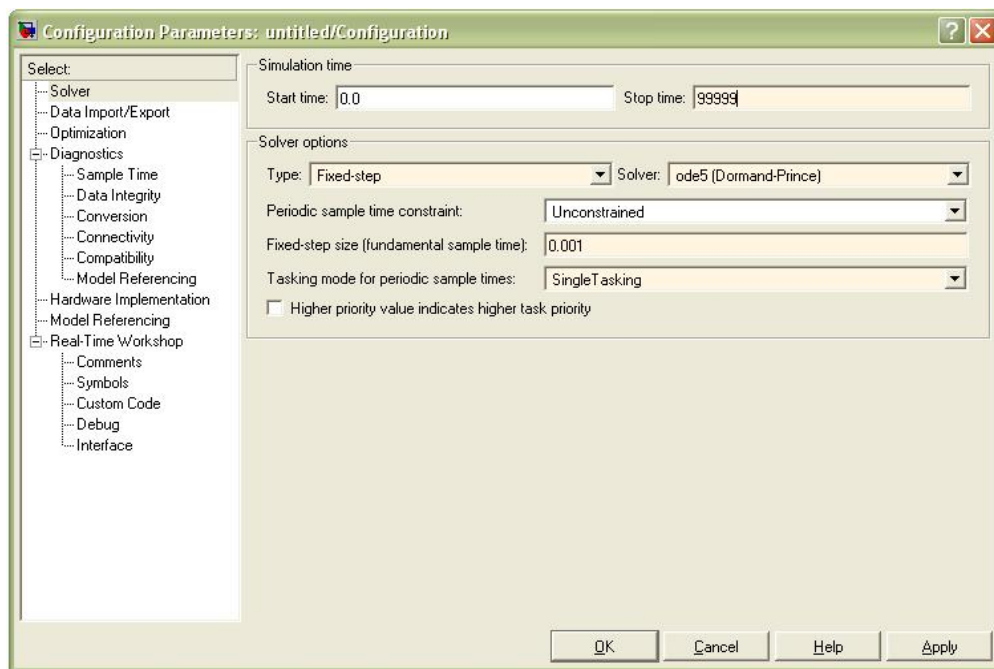


Figure 3.10: Configuration Parameters – Solver

7. One of following is done:
  - **Apply** is clicked to apply the changes to the model and the dialog box is left open.
  - **OK** is clicked to apply the changes to the model and the **Configuration Parameters** dialog box is closed.

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

The Simulation Parameters are used by Real-Time Workshop for generating C code and building a real-time application.

1. In the Simulink window, **Configuration Parameters** is clicked from the **Simulation** menu as shown in Figure 3.19.
2. The **Hardware Implementation** node is clicked.
3. From the **Device type** list, 32-bit Real-Time Windows Target is chosen.

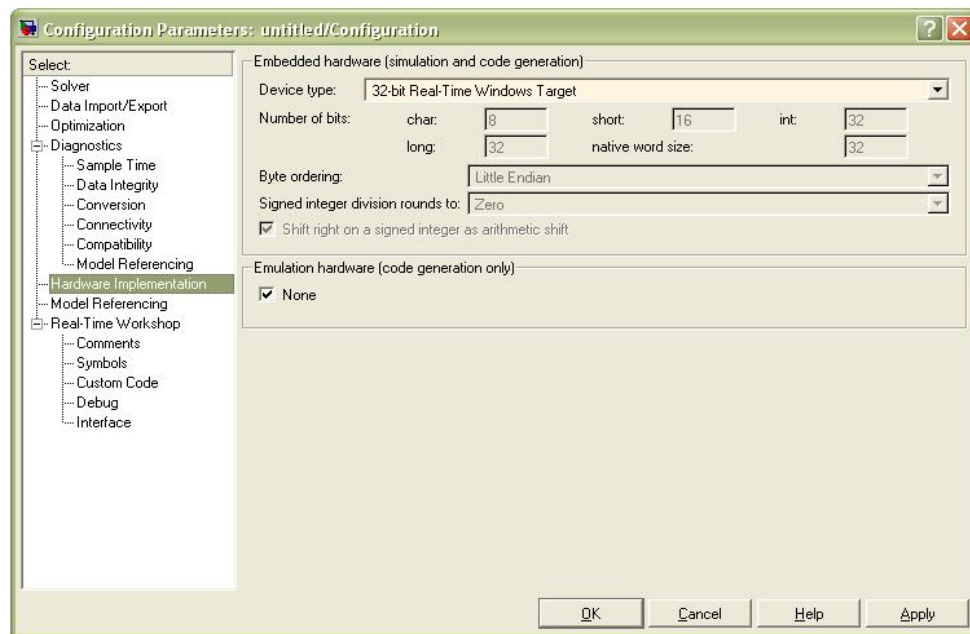


Figure 3.11: Configuration Parameters – Hardware Implementation

4. The **Real-Time Workshop** node is clicked.  
The **Real-Time Workshop** pane will open.
4. In the **Target selection section**, the **Browse** button is clicked at the **RTW system target file** list. The **System Target File Browser** will open as shown in Figure 3.20.
5. The system target file is selected for the Real-Time Windows Target and **OK** is clicked.

proosek.tlc	OSEK Target for 3Soft ProOSEK Implementat
rsim.tlc	Rapid Simulation Target
rtwin.tlc	Real-Time Windows Target
rtwsfcn.tlc	S-function Target
ti_c2000_ert.tlc	Embedded Target for TI C2000 DSP (ERT)
ti_c2000_grt.tlc	Embedded Target for TI C2000 DSP (GRT)

Figure 3.12: System Target File Browser

The system target file `rtwin.tlc`, the template makefile `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 after **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 would look similar to the Figure 3.21.

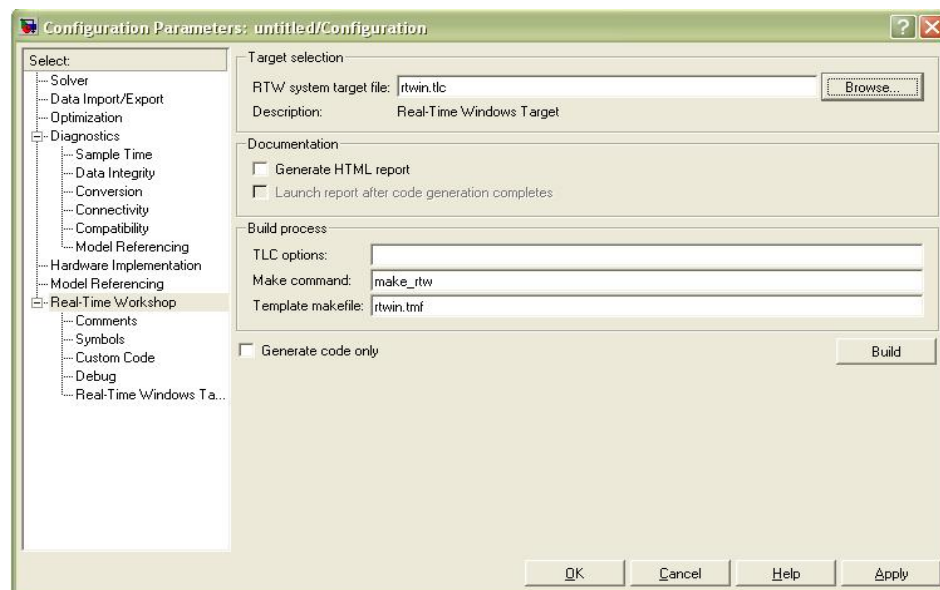


Figure 3.13: Configuration Parameters – Real-Time Workshop

- One of following is done:

- **Apply** is clicked to apply the changes to the model and the dialog box is left open.
- **OK** is clicked to apply the changes to the model and the **Configuration Parameters** dialog box is closed.

### 3.1.3.4 Creating a Real-Time Application

Real-Time Workshop generates C code from the Simulink model and then the **Microsoft Visual Basic C++** compiler compiles and links that C code into a real-time application. After parameters are entered into the **Configuration Parameters** dialog box for Real-Time Workshop, a real-time application could be built.

1. In the Simulink window and from the Tools menu, it should be pointed to the Real-Time Workshop and then clicked Build Model. 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 makefile `rtwin_model.mk` from the template makefile `rtwin.tmf`.
  - The make utility `make_rtw.exe` builds the real-time application `rtwin_model.rwd` using the makefile `rtwin_model.mk` created above. The file `rtwin_model.rwd` is binary files that refer to as the real-time application. The real-time application could be run with the Real-Time Windows Target kernel.
2. The Simulink model is connected to real-time application.  
 After the real-time application is created, MATLAB could be closed and started again later and then the executable is connected and run without having to rebuild.


### 3.1.3.5 Running a Real-Time Application

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

The process of connecting consist of

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

After the real-time application is built, the model could be run in real time.

1. From the Simulation menu, External is clicked and then Connect To Target is connected from the Simulation menu, Also, it could be connected to the target from the toolbar by clicking . It can be seen I Figure 3.22.

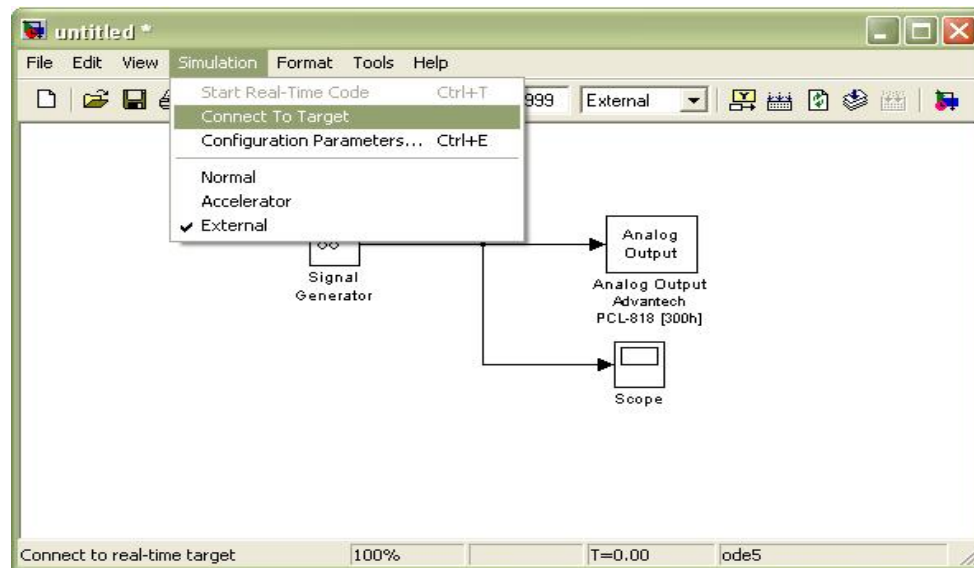


Figure 3.14: Connect to target from the Simulation menu

MATLAB will display the message  
Model rtwin\_model loaded

2. In the Simulation window and from the **Simulation** menu, **Start Real-Time Code** is clicked. The execution also could be started from the toolbar by clicking Start icon.

Simulink runs the execution and plots the signal data in the Scope window.

In the model, 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 output 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 is let to be run until it reaches the stop time.
  - **Stop Real-Time Code** is clicked from the **Simulation** menu.

The real-time application is stopped.

4. In the Simulation window, **Disconnected From Target** is clicked from the **Simulation** menu.
5. From the **Simulation** menu, **External** is clicked  
MATLAB will display the message  
Model rtwin\_model unloaded

### 3.1.4 Creating Graphical User Interfaces

MATLAB implements GUIs as figure windows containing various styles of uicontrol objects. You must program each object to perform the intended action when activated by the user of the GUI. In addition, you must be able to save and launch your GUI. All of these tasks are simplified by GUIDE, MATLAB's graphical user interface development environment.

### 3.1.4.1 GUI Development Environment

The process of implementing a GUI involves two basic tasks:

- (i) Laying out the GUI components
- (ii) Programming the GUI components

GUIDE primarily is a set of layout tools. However, GUIDE also generates an M-file that contains code to handle the initialization and launching of the GUI. This M-file provides a framework for the implementation of the *callbacks* – the functions that execute when users activate components in the GUI.

#### The Implementation of a GUI

While it is possible to write an M-file that contains all the commands to lay out a GUI, it is easier to use GUIDE to lay out the components interactively and to generate two files that save and launch the GUI:

- (i) A FIG-file – contains a complete description of the GUI figure and all of its children (uicontrols and axes), as well as the values of all object properties.
- (ii) An M-file – contains the functions that launch and control the GUI and the callbacks, which are defined as subfunctions. This M-file is referred to as the *application M-file* in this documentation.

### 3.1.4.2 Starting Guide

Start GUIDE by typing `guide` at the MATLAB command prompt. This displays the **GUIDE Quick Start** dialog, as shown in the following Figure 3.23.

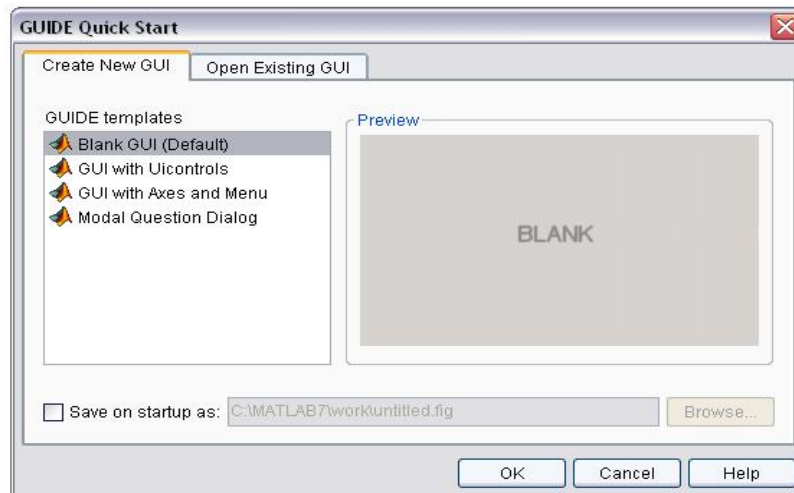


Figure 3.15: GUIDE Quick Start

From the Quick Start dialog, the user can:

- (i) Create a new GUI from one of the GUIDE templates.
- (ii) Open an existing GUI.

### 3.1.4.3 The Layout Editor

When the user opened a GUI in GUIDE, it is displayed in the Layout Editor, which is the control panel for all of the GUIDE tools. The following Figure 3.24 shows the Layout Editor with a blank GUI template.

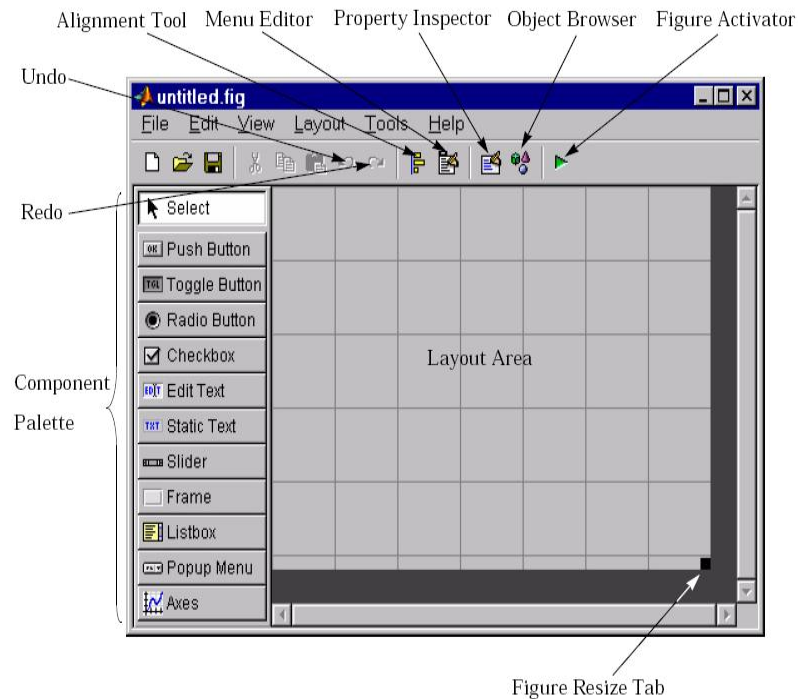


Figure 3.16: Layout Editor

The user can lay out the GUI by dragging components, such as panels, push buttons, pop-up menus, or axes, from the component palette, at the left side of the Layout Editor, into the layout area.

#### 3.1.4.4 Programming a GUI

After laying out the GUI and setting component properties, the next step is to program the GUI. The user programs the GUI by coding one or more callbacks for each of its components. Callbacks are functions that execute in response to some action by the user. A typical action is clicking a push button.

A GUI's callbacks are found in an M-file that GUIDE generates automatically.

GUIDE adds templates for the most commonly used callbacks to this M-file, but the user may want to add others. Use the M-file Editor to edit this file.

The following Figure 3.25 shows the Callback template for a push button.

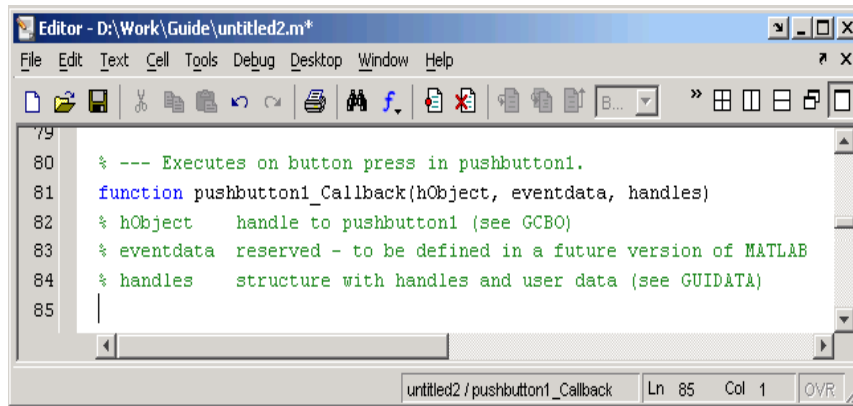


Figure 3.17: M-file Editor

### 3.1.5 Simulation of the servomotor

To know or improve if the gain value that we get before is correct or not, build the simulink model using Matlab. By this simulink, it will show the result in the feedback system by the graph of the comparison of feedback value and input value. If the value of feedback after the gain equal to the input value then our value is correct.

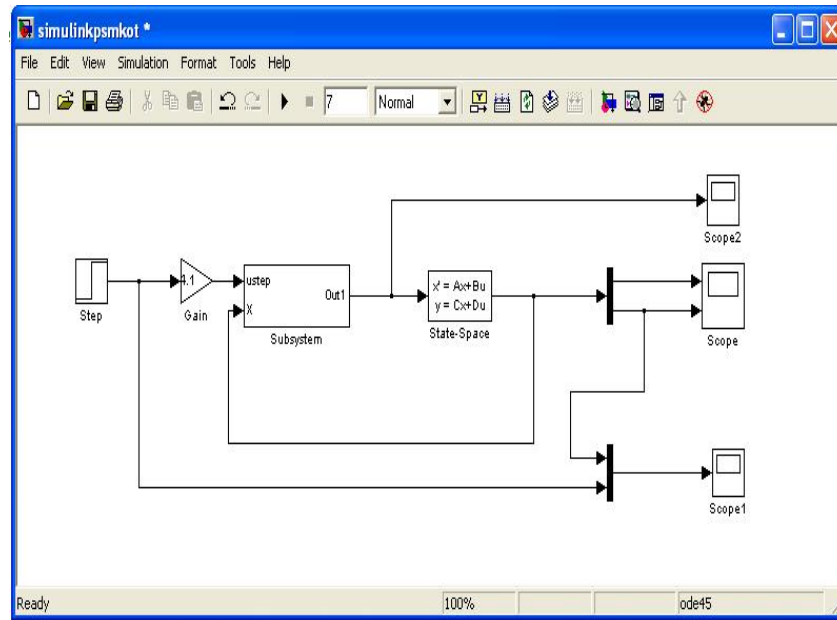


Figure 3.18: Block diagram for simulation servo motor

There is subsystem in the main system to make it easy to build the system. In the subsystem, there are systems that are build connected to the main system.

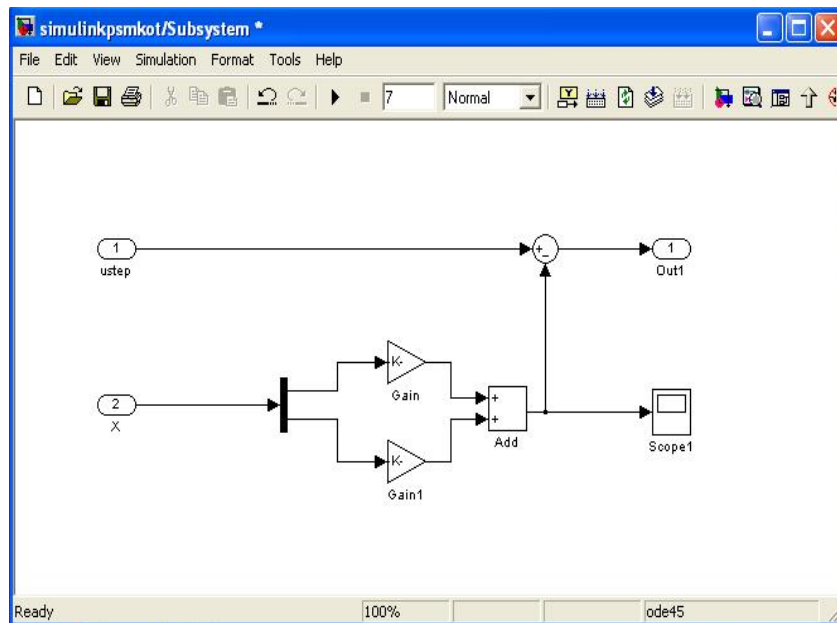
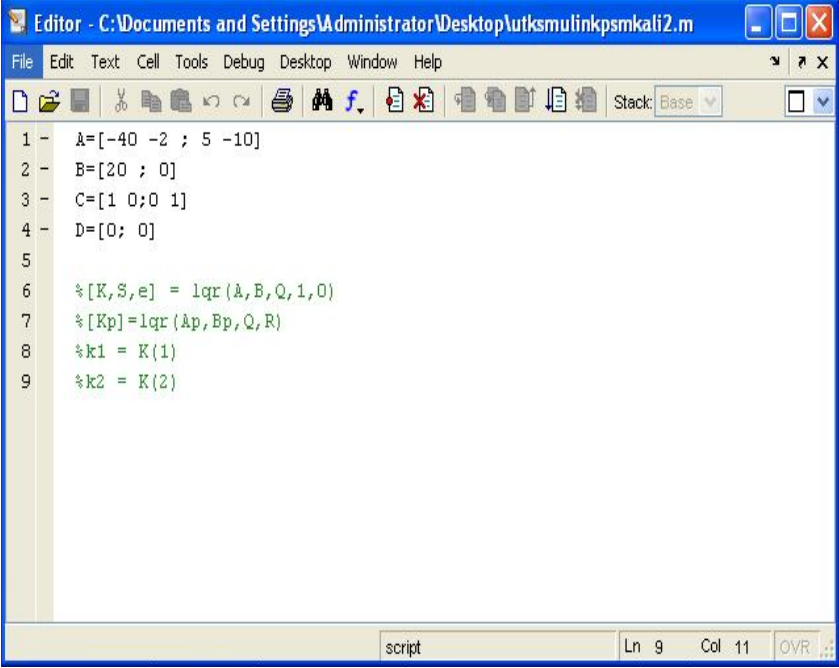


Figure 3.19: Subsystem for the block diagram above

Before the simulink above run, the m-file that are contains the value of each parameters in the block diagram need to run first. The m-file is:



```

1 - A=[-40 -2 ; 5 -10]
2 - B=[20 ; 0]
3 - C=[1 0;0 1]
4 - D=[0; 0]
5
6 - % [K,S,e] = lqr(A,B,Q,1,0)
7 - % [Kp]=lqr(Ap,Bp,Q,R)
8 - % k1 = K(1)
9 - % k2 = K(2)

```

The image shows a MATLAB Editor window titled 'Editor - C:\Documents and Settings\Administrator\Desktop\lqtsmulinkpsmkali2.m'. The window contains a script with 9 lines of code. Lines 1-4 define matrices A, B, C, and D. Line 5 is a blank line. Lines 6-9 are comments for the LQR function call and its outputs. The status bar at the bottom indicates 'script', 'Ln 9', 'Col 11', and 'OVR'.

Figure 3.20: M-file for the system

### 3.1.6 Graphical User Interface

User interface is developing to make the user controlling the speed of the motor easily by just change the value in the text box. The main panel of the user interface is:

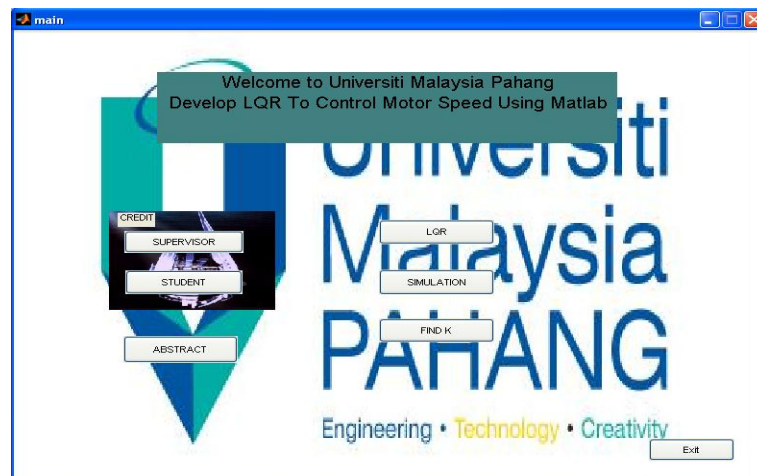


Figure 3.21: Main panel

From the main, user can go to the other subsystem for GUI. There's credit for student and supervisor of the project, abstract of the project, 3 buttons for the LQR, Simulation and Find K, and button exit.

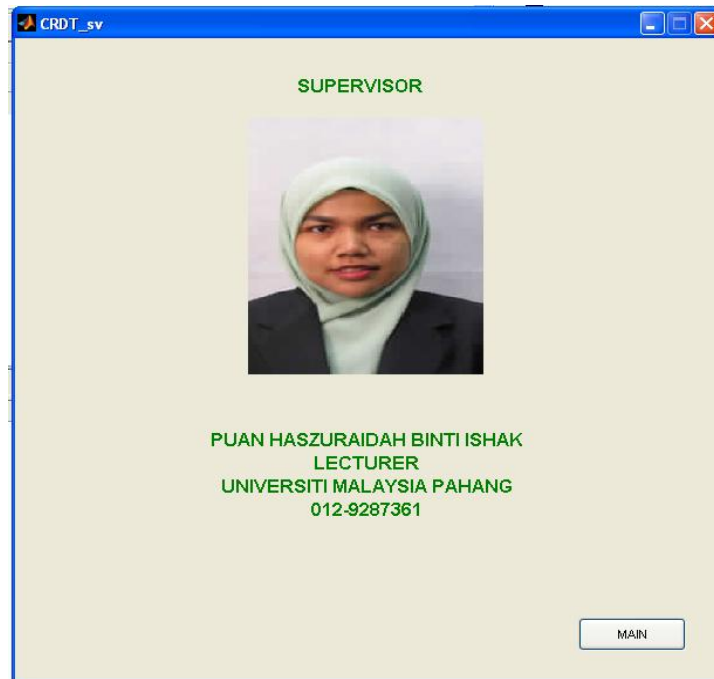


Figure 3.22: Supervisor credit

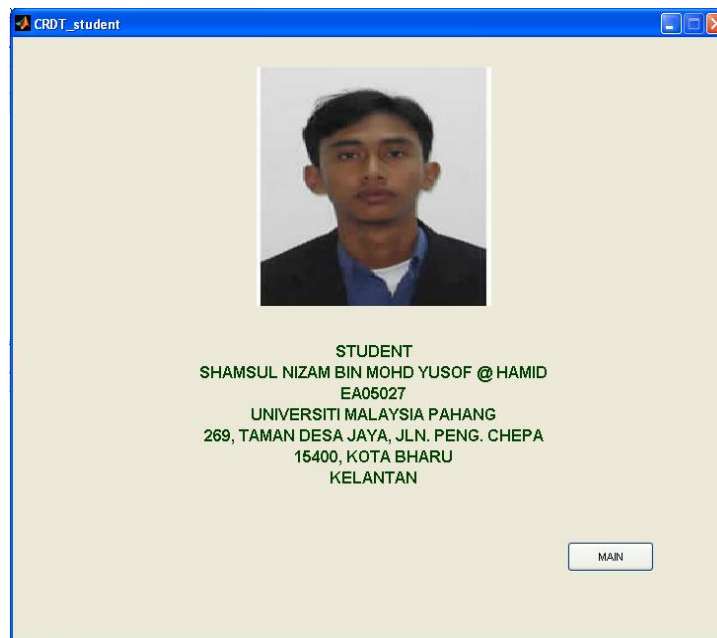


Figure 3.23: Student credit

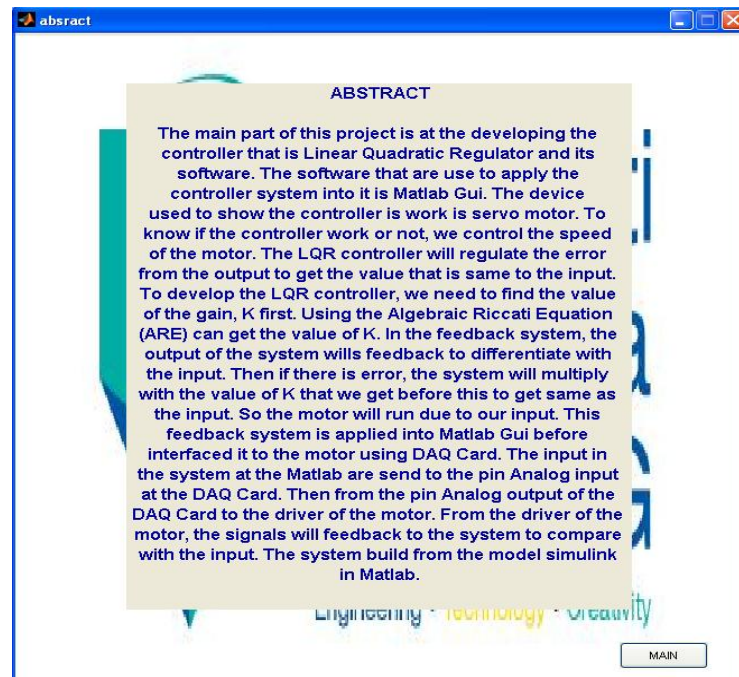


Figure 3.24: Abstract of the project

For the button LQR on the main panel, the following figure will show the m-file appears when the button pressed.

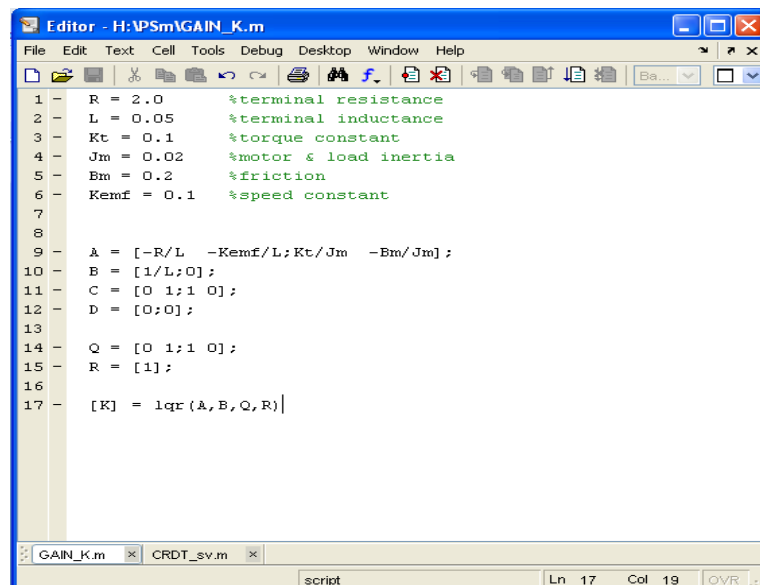


Figure 3.25: Manual way to find the K gain

After the m-file ran, we click the button Simulation so that we can simulate the system with the value of K obtained before. Then after the simulation done, click on the scope to sure if the result is perfect. From the result, we can know if the value of K is correct. The simulation button pressed then the model will appear like below:

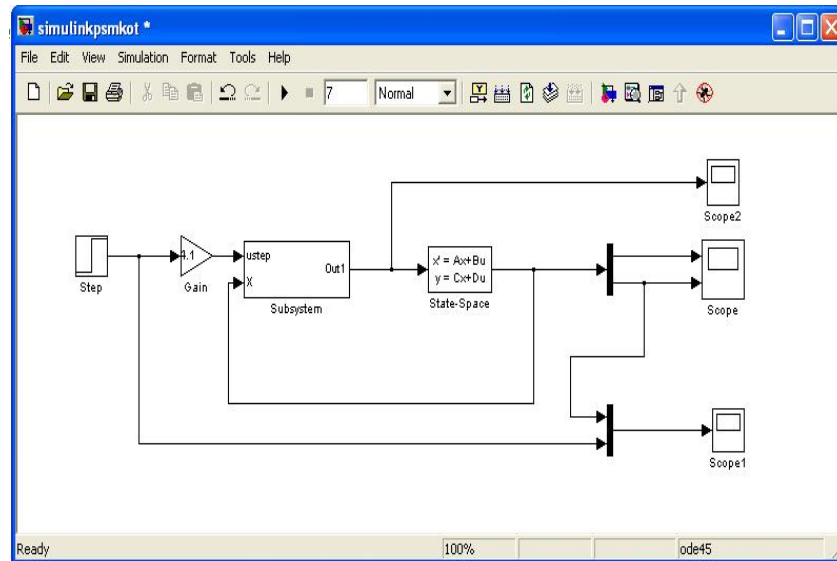


Figure 3.26: Simulink model

There also GUI for finding K for just key in the value of the motor constant because different type of motor, it has different constant. The value of K will show up in the 2 text boxes in form K1 and K2. The GUI is:

Figure 3.27: Find K if different motor constant

## 3.2 HARDWARE

### 3.2.1 CLIFTON PRECISION SERVO MOTOR MODEL JDH-2250-HF-2C-E

The other advantages of servo motor are:

- High output power relative to motor size and weight
- Encoder determines accuracy and resolution
- High efficiency. Can approach 90% at light loads
- High torque to inertia ratio. Can rapidly accelerate loads
- Has reserve power. Two to three times continuous power for short periods
- Has reserve torque. Five to ten times rated torque for short periods
- Motor stays cool. Currently draw proportional to load
- Usable high speed torque. Maintains rated torque to 90% of No load RPM
- Audibly quiet at high speeds
- Resonance and vibration free operation



Figure 3.28: Servo Motor

The parameters of the Clifton Servo Motor

- Torque Constant: 15.76 oz-in. / A
- Back EMF: 11.65 VDC / KRPM
- Peak Torque: 125 oz-in.



The several characteristics of Servo Drive G340 are:

- Run PM DC servos with stepmotor software
- PLL step pulse multiplier
- 20A motor output
- 18V to 80V power supply
- PID closed loop operation
- Step and Direction control inputs
- Quadrature encoder feedback
- Anti-dithering circuit keeps motor silent
- Onboard +5VDC, 50mA encoder supply
- Pulse by pulse motor current limiting
- 0 to 20A current limit adjust range
- +/- 128 count servo lock range
- No tachometer feedback needed
- 250 kHz max step rate
- Latched Fault protection
- LED Fault indicator
- Small size: 2.5" by 2.5" by 0.82" (63mm by 63mm by 21mm)
- Light: 3.6 oz (100gm)
- Anodized aluminum package
- Modular 2-piece main connector

### **3.2.3 G340 installation**

#### **3.2.3.1 Encoder hook up**

The instructions below are to determine the optimal encoder line count:-

- i. Determine motors no load RPM

- ii. Calculate rated RPM as 80% of no load RPM
- iii. Divide (#2) by 60 to get revolutions per second
- iv. Determine the CNC program's maximum step pulse frequency (in Hz)
- v. Divide (#4) by (#3), which will give you the maximum counts per revolution
- vi. Divide (#5) by 4, which will give you the max line count
- vii. Pick the first standard line count below (#6)

An example of using the formula with 45 kHz step pulse frequency and a maximum motor RPM of 3000:

$$(45 \text{ kHz} / 40) / 4 = 281.25$$

### **3.2.3.2 Power supply hook up**

Power supply needs to keep leads short and the largest wire gauge used so that it will fit in the terminals. Use a 1000  $\mu$ F capacitor across the G340 power supply terminals if the lead length is more than 18". Make sure the power supply can provide the peak current the motor may draw. The range for the power supply is between 18 VDC to 80 VDC. The actual voltage should not higher than the motor's rated voltage that is 5 volts.

### **3.2.3.3 Testing the encoder**

Use oscilloscope or voltmeter to monitor the POSITION ERROR test point. The POSITION ERROR test point shows the difference between the command position and the actual motor position. If both same, the voltage will be +5 VDC.

### 3.2.3.4 Control input hook up

The control input group is the standard step motor drive STEP, DIRECTION and +5 VDC lines. The STEP and DIRECTION signal drivers must be TTL compatible and have edge transition times of 100 ns or faster. The +5 VDC is the opto-isolator common anode line and must be returned to the pulse source +5 VDC supply.

- i. **DIR** Connect the DIRECTION line to this terminal
- ii. **STEP** Connect the STEP line to this terminal
- iii. **+5 VDC** Connect this terminal to the controller +5 VDC power supply

### 3.2.3.5 Testing the control inputs

To testing the functionality of these output, let the oscillator used before at the same position. Then, set the STEP pulse generator to about 40 pulses per second and set the DIRECTION output to clockwise (logical “1”). Power supply turned on. After the power-on reset period of 5 seconds the FAULT light will turn off.

### 3.2.3.6 Motor hook up

Make sure the power is off and the STEP pulse source is set to zero pulses per second. The trimpot settings are checked if it is set to according to the instructions above. Secure the motor so that it can't jump off the bench.

**ARM-** Connect the BLACK motor lead to this terminal

**ARM+** Connect the RED motor lead to this terminal

### 3.2.4 Advantech PCI-1710HG

The Advantech PCL-1710HG is the perfect choice to use for this project because it is low cost. The budget will be saving to buy this multifunction DAS (Data Acquisition System) card. It present in the best price and performance in the market. This custom gives higher performance and reliability with lower power consumption. The size of this hardware is half size DAS Card. The software is compatible for this PCL from MATLAB. Figure 3.7 shows the Advantech PCI-1710HG.

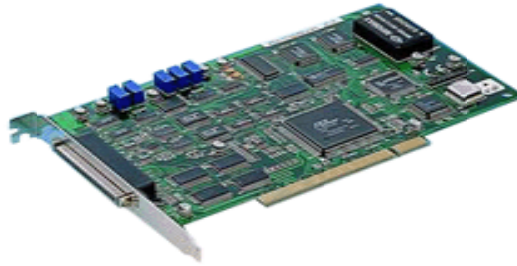


Figure 3.31: Advantech PCI-1710HG

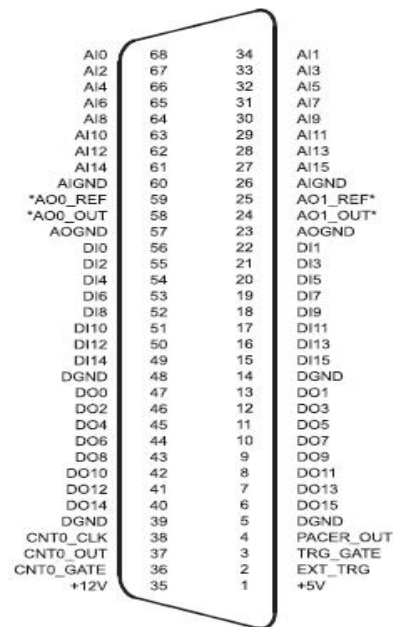
### 3.2.5 Common Specifications

- a) Analog Input
  - Channels: 16, single-ended or 8 differential.
  - Resolution: 12 bits
  - Max. Sampling Rate: 100kS/s
  - Input range selection: (V, software programmable)
  - Auto channel/gain scanning
  - Input impedance: 1 G ohms
  - Input overvoltage: +/-30 VDC maximum
- b) Analog Output
  - Channels: 2

- Resolution: 12-bits
- Output range: (V, software programmable)

### 3.2.6 Pin Assignments

Figure 3.8 shows the Pin Assignment for PCI-1710 HG. It used to connect from the I/O board to PC.



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

Figure 3.32: Pin Assignment for PCI-1710 HG

## 3.3 Linear Quadratic Regulator Development

### 3.3.1 The General State-Space Representation

A physical network in state space has been represented and has a good idea of the terminology and the concept, let's summarize and generalize the representation for linear differential equations. First, formalize some of the definitions that came across before.

*Linear combination:* A linear combination of  $n$  variables,  $x_i$ , for  $i = 1$  to  $n$ , is given by the following sum,  $S$ :

$$S = K_n x_n + K_{n-1} x_{n-1} + \dots + K_1 x_1 \quad (3.1)$$

where each  $K_i$  is a constant.

*Linear independence:* A set of variables is said to be linearly independent if none of the variables can be written as a linear combination of the others. For example, given  $x_1$ ,  $x_2$ , and  $x_3$ , if  $x_2 = 5x_1 + 6x_3$ , then the variables are not linear combination of the other two. Now, what must be true so that one variable cannot be written as a linear combination of the other variables? Consider the example  $K_2 x_2 = K_1 x_1 + K_3 x_3$ . If no  $x_i = 0$ , then any  $x_i$  can be written as a linear combination of other variables, unless all  $K_i = 0$ . Formally, then, variables  $x_i$ , for  $i = 1$  to  $n$ , are said to be linearly independent if their linear combination,  $S$ , equals zero only if every  $K_i = 0$  and no  $x_i = 0$ .

*System variable:* Any variable that responds to an input or initial conditions in a system.

*State variables:* The smallest set of linearly independent system variables Such that the values of the members of the set at time  $t_0$  along with known forcing functions completely determine the value of all system variables for all  $t \geq t_0$ .

*State vector:* A vector whose elements are the state variables.

*State space:* The  $n$ -dimensional space whose axes are the state variables. This is a new term and is illustrated in Figure 3.1, where the state variables are assumed to be a resistor voltage,  $v_R$ , and a capacitor voltage,  $v_C$ . These variables form the axes of the *state space*. A trajectory can be thought of as being mapped out by the state vector,  $\mathbf{x}(t)$ , for a range of  $t$ . Also shown is the state vector at the particular time  $t = 4$ .

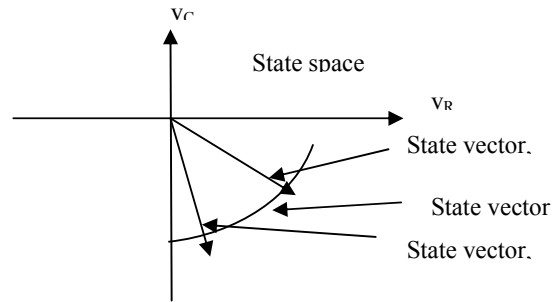


Figure 3.33: Graphic representation of state space and a state vector

*State equations:* A set of simultaneous, first-order differential equations with  $n$  variables, where the  $n$  variables to be solved are the state variables.

*Output equation:* The algebraic equation that expresses the output variables of a system as linear combinations of the state variables and the inputs.

Now that the definitions have been formally stated, define the state-space representation of a system. A system is represented in state space by the following equations:

$$\begin{aligned}\dot{x} &= Ax + Bu \\ y &= Cx + Du\end{aligned}$$

for  $t \geq t_0$  and initial conditions,  $x(t_0)$ , where

$x$  = state vector

$\dot{x}$  = derivative of the state vector with respect to time

$y$  = output vector

$u$  = input or control vector

$A$  = system matrix

$B$  = input matrix

$C$  = output matrix

$D$  = feedforward matrix

Equation (1) above is called the state equation and the vector  $x$ , the state vector, contains the state variables. That equation can be solved for the state variables. Equation (2) is called output equation. This equation used to calculate any other system variables. This representation of a system provides complete knowledge of all variables of the system at any  $t \geq t_0$ .

### 3.3.2 Evaluating the State Space equation

The main point for this project is LQR. We need to obtain the gain,  $K$  for the linear state feedback control. First, the state space equation of the servo motor need to be obtain because each motor have different state space equation. State space for the servo motor model calculated below:-

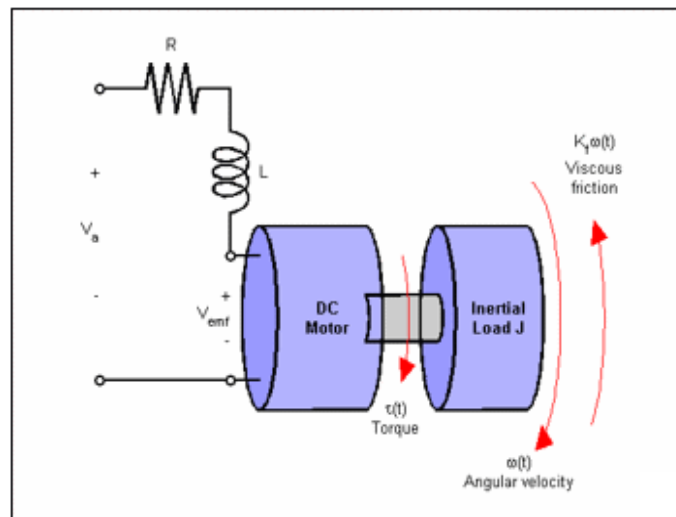


Figure 3.34: DC motor model system

where

$R$  = terminal resistance

$L$  = terminal inductance

$K_t$  = torque constant

$J_m$  = motor and load inertia

$B_m$  = friction

$K_{emf}$  = speed constant

Using Kirchoff current law;

$$\begin{aligned}
 V_s(t) &= V_{emf} + L \frac{di}{dt} + Ri(t) & ; V_{emf} &= K \omega \\
 V_s(t) &= K\omega + L \frac{di}{dt} + Ri(t) \\
 L \frac{di}{dt} &= V_s(t) - K\omega - Ri(t) \\
 \frac{di}{dt} &= \{V_s(t) - K\omega - Ri(t)\}/L
 \end{aligned} \tag{4.1}$$

Torque,

$$\begin{aligned}
 T &= Kt.Ii \\
 T &= J \frac{d\omega}{dt} + B\omega \\
 Kt.Ii &= J \frac{d\omega}{dt} + B\omega \\
 J \frac{d\omega}{dt} &= Kt.Ii - B\omega \\
 \frac{d\omega}{dt} &= \{Kt.Ii - B\omega\}/J
 \end{aligned} \tag{4.2}$$

Simplify equation 4.1 & 4.2

$$\frac{di}{dt} = -\frac{R}{L} \cdot i(t) - \frac{K\omega}{L} + \frac{1}{L} V_s(t) \tag{4.3}$$

$$\frac{d\omega}{dt} = \frac{Kt}{J} \cdot i(t) - \frac{B}{J} \cdot \omega \tag{4.4}$$

After get all the equation above, all the equation arrange and insert into the state space equation in matrix form. The equation of the state space is:-

$$\begin{aligned}
 \dot{x} &= Ax + Bu \\
 y &= Cx + Du
 \end{aligned}$$

$$\begin{pmatrix} \frac{di}{dt} \\ \frac{d\omega}{dt} \end{pmatrix} = \begin{bmatrix} -\frac{R}{L} & -\frac{K}{L} \\ \frac{Kt}{J} & -\frac{B}{J} \end{bmatrix} \begin{bmatrix} i \\ \omega \end{bmatrix} + \begin{bmatrix} \frac{1}{L} \\ 0 \end{bmatrix} V_s(t)$$

$$V_{emf} = [0 \ 1] \begin{bmatrix} i \\ \omega \end{bmatrix}$$

where;

$$A = \begin{bmatrix} -\frac{R}{L} & -\frac{K_{emf}}{L} \\ \frac{K_t}{J} & -\frac{B}{J} \end{bmatrix}, \quad C = [0 \ 1]$$

$$B = \begin{bmatrix} \frac{1}{L} \\ 0 \end{bmatrix}, \quad D = 0$$

Then, get the elements from the DC motor

Table 3.2: Parameters of servomotor elements

<i>Variable</i>	<i>Units</i>	<i>Value</i>	<i>Description</i>
R	Ohm	2.7	Terminal Resistance
L	Henry	0.004	Terminal Inductance
K <sub>t</sub>	N-m-	0.105	Torque Constant
K <sub>emf</sub>	V-s-	0.105	Back <i>emf</i> Constant
B	N-m-s-	0.0000093	Friction
J	Kg-	0.0001	Equivalent Moment of Inertia
T <sub>d</sub>	-	A=0.05N-m , f=100 rad-	Load Disturbance

Convert to state space equation:

$$\begin{pmatrix} \frac{di}{dt} \\ \frac{d\omega}{dt} \end{pmatrix} = \begin{bmatrix} -\frac{R}{L} & -\frac{K_{emf}}{L} \\ \frac{K_t}{J} & -\frac{B}{J} \end{bmatrix} \begin{bmatrix} i \\ \omega \end{bmatrix} + \begin{bmatrix} \frac{1}{L} \\ 0 \end{bmatrix} V_s(t)$$

Thus, the perimeter is state below:

$$A = \begin{bmatrix} -\frac{R}{L} & -\frac{K_{emf}}{L} \\ \frac{K_t}{J} & -\frac{B}{J} \end{bmatrix} = \begin{bmatrix} -\frac{2.7}{0.004} & -\frac{0.105}{0.004} \\ \frac{0.105}{0.0001} & -\frac{0.0000093}{0.0001} \end{bmatrix} = \begin{bmatrix} -675 & -26.25 \\ 1050 & -0.093 \end{bmatrix}$$

$$B = \begin{bmatrix} \frac{1}{L} \\ 0 \end{bmatrix} = \begin{bmatrix} \frac{1}{0.004} \\ 0 \end{bmatrix} = \begin{bmatrix} 250 \\ 0 \end{bmatrix}$$

$$C = [0 \ 1]$$

$$D = 0$$

### 3.4 Applying Linear Quadratic Regulator

The plant is defined by the following state space equations as state in DC motor equation:

$$\begin{aligned}\dot{x} &= Ax + Bu \\ y &= Cx + Du\end{aligned}$$

where :

$$A = \begin{bmatrix} -4 & -0.2 \\ 5 & -10 \end{bmatrix}, \quad B = \begin{bmatrix} 2 \\ 0 \end{bmatrix}, \quad C = [0 \ 1] \quad D = 0$$

The performance index J is given by

$$J = \int_0^{\infty} (x^* Q x + u^* R u) dt$$

where

$$Q = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \quad R = 1$$

Assume the following control  $u$  is used.

$$u = -Kx$$

Determine the optimal feedback gain matrix K.

The optimal feedback gain matrix K can be obtained by solving the following Riccati equation for a positive-definite matrix P.

$$A * P + PA - PBR^{-1}B * P + Q = 0$$

The result is

$$\mathbf{P} = \begin{bmatrix} 0.1349 & 0.0152 \\ 0.0152 & 0.0496 \end{bmatrix}$$

Substituting this matrix P into the following equation gives the optimal K matrix:

$$K = R^{-1}B'P$$

$$= [1][0 \ 2] \begin{bmatrix} 0.1349 & 0.0152 \\ 0.0152 & 0.0496 \end{bmatrix}$$

$$= [0.2698 \ 0.0304]$$

Thus, the optimal control signal is given by

$$u = -Kx = -x_1 - x_2$$

## **CHAPTER 4**

### **RESULTS AND DISCUSSION**

#### **4.1 Simulation result**

For the simulation, the parameters of DC motor and servo motor will be taken to show the result of the simulation.

##### **4.1.1 Simulation on the DC motor**

The result is for the DC motor that has the value of parameters below:

$$R = 2.0 \, \Omega$$

$$L = 0.05 \, \text{H}$$

$$K_t = 0.1$$

$$J_m = 0.02$$

$$B_m = 0.2$$

$$K_{emf} = 0.1$$

The value of the gain,  $K$  obtained from the GUI Find  $K$  after all the parameters of the DC motor selected inserted. The figure shows the result from the GUI:

The screenshot shows a MATLAB GUI titled 'FIND\_K'. It contains several input fields for motor parameters:  $L_a$  (0.05),  $R_a$  (2.0),  $D_m$  (0.2),  $K_{emf}$  (0.1),  $K_t$  (0.1), and  $J$  (0.02). A central text box instructs the user to 'Insert the constant value from the DC motor to get the value of gain before proceed to the LQR controller'. To the right of the text is an image of a DC motor. Below the parameters, there are two output fields for 'Gain K': 0.245006 and 0.0801054. At the bottom, there are buttons for 'RUN', 'CLEAR', 'MAIN', and 'NEXT'.

Figure 4.1: Result for DC motor

From the figure above, the value of the gain,  $K$  has been obtained after all the parameters of the DC motor inserted and the button RUN clicked. The value is:

$$K = [0.245006 \ 0.0801054]$$

Using the gain value that has been obtained before, the simulation is done by inserting the value into the block diagram system.

For the simulation result in Matlab, the graph is like below:

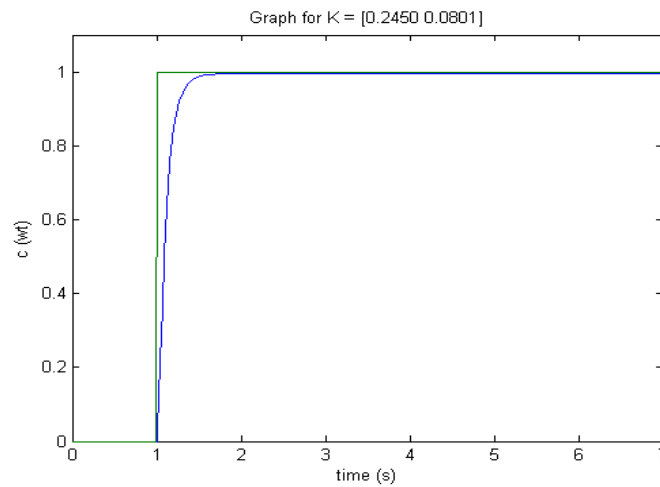


Figure 4.2: Graph result for  $K = [0.245006 \ 0.0801054]$

#### 4.1.2 Simulation on the Clifton Precision servo motor

To find the other value of gain, K for Clifton Precision servo motor, click the button Clear on the user interface and insert the value of the Clifton Precision servo motor parameters. The parameters values of the Clifton Precision servo motor are like below:

$$R = 2.7$$

$$L = 0.004$$

$$K_t = 0.105$$

$$J_m = 0.0001$$

$$B_m = 0.0000093$$

$$K_{emf} = 0.105$$

The following figure shows the result of the value of gain, K obtained from the user interface before. The value of gain, K obtained is:

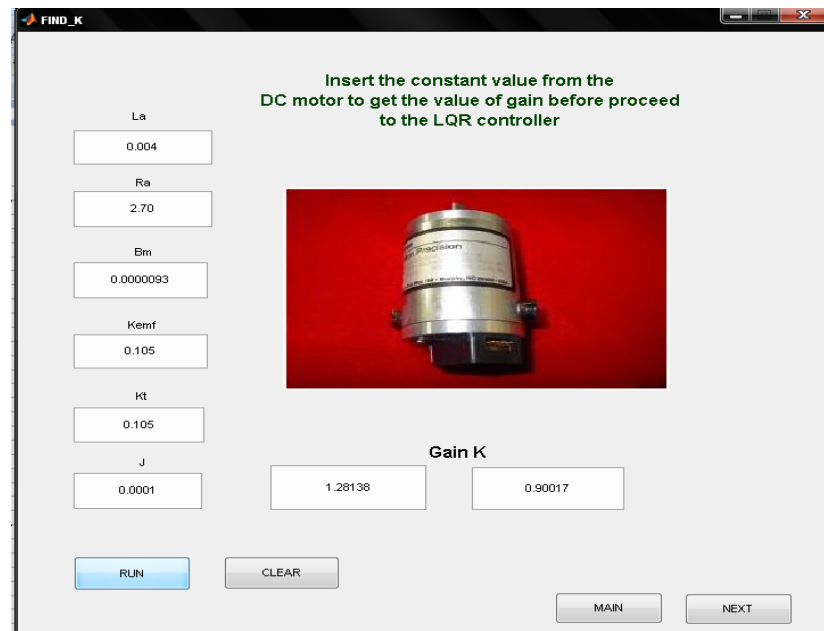


Figure 4.3: Result for Clifton Precision servo motor

The result of the gain, K in matrix form from figure above is like below:

$$K = [1.28138 \ 0.90017]$$

Then, insert the value that has been obtained before into the simulink system.  
After the simulink run, the graph of the result is:

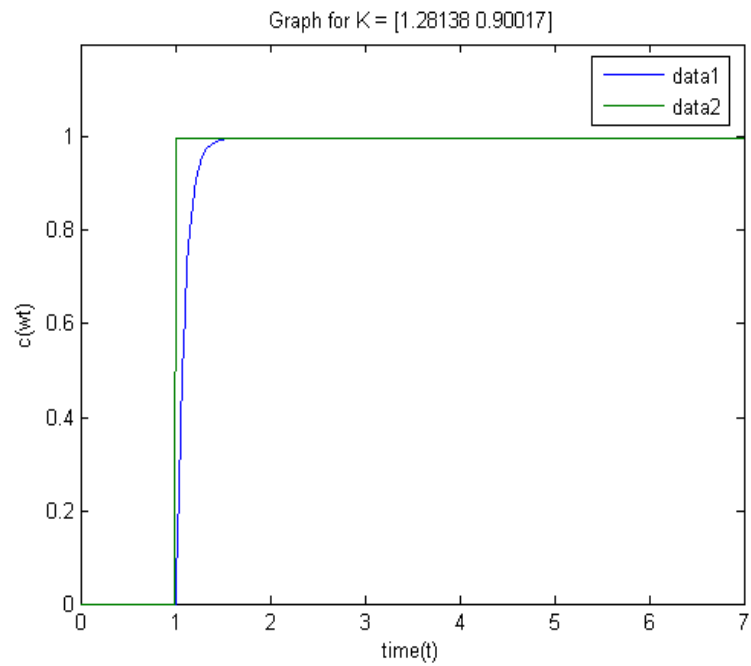


Figure 4.4: Graph result for K = [1.28138 0.90017]

## 4.2 Data Analysis

Analysis will be done on the first simulation because analysis for the second simulation that is simulation for the Clifton Precision is the same as the first one.

From the graph result of the first simulation, DC motor simulation, the data analysis is get from the following figure. For the first figure below, Settling time ( $T_r$ ) of the system get stable:

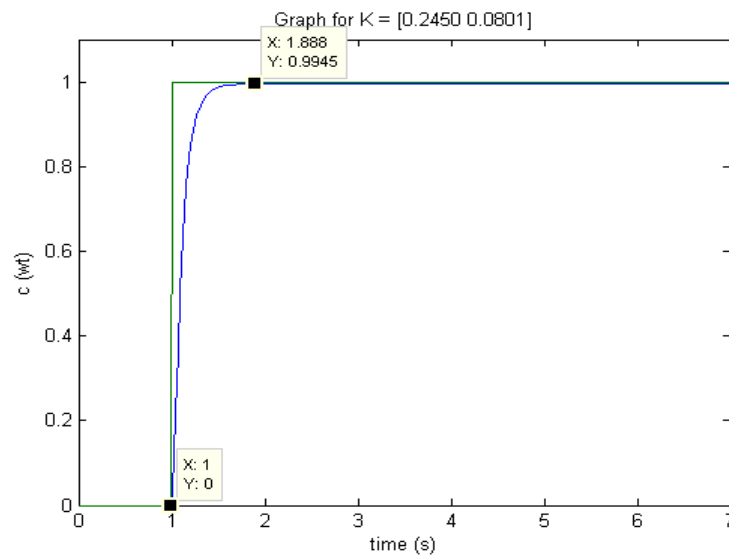


Figure 4.5: Settling time for simulation DC motor

From the graph above, the settling time for the system to stable state are:

$$\begin{aligned} T_s &= 1.888 - 1 \\ &= 0.888s \end{aligned}$$

For the next figure, the rise time will be analyzed from the graph shown below:

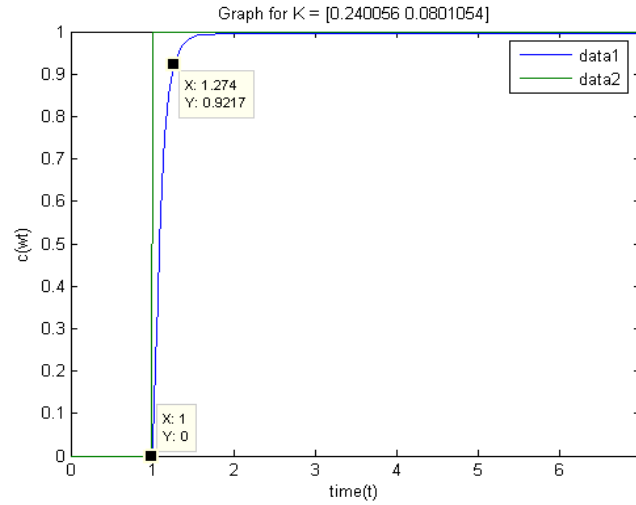


Figure 4.6: Rise time for simulation DC motor

For the figure above, analysis can be done for the rise time. From the graph, the time for the system to be rise to the steady state is:

$$\begin{aligned} Tr &= 1.274 - 1 \\ &= 0.274 \end{aligned}$$

From the two figures above, analysis also can be done to the other parameters like Percent Overshoot (%OS), and Steady state error. Table below shows the all parameters from the analysis that has been done before for the first simulation and the parameters mentioned earlier. All the parameters are put in the table below:

Table 4.1: Data analysis simulink of the system

Rising time (Tr)	0.274s
Settling time (Ts)	0.888
Percent overshoot %OS	0
Steady state error	0

From the table of the parameters analysis, the system gets stable in a short time, within a second. The system that use LQR controller also does not have the Percent Overshoot (%OS) and Steady state error. This can be seen in the figure above that the graph is rising smoothly.

### 4.3 Tuning the value of Q

The value of Q can be tuning to get the best result of the LQR as long as the value Q is positive semi definite. For the analysis before, the value of Q is 1, same as the R. Here, the value of Q will be used as 10 and 100 but the value of R is still maintained.

For the first tuning, the value of Q is 1 will be used. The result that gets from the Matlab after tuning the value of Q is:

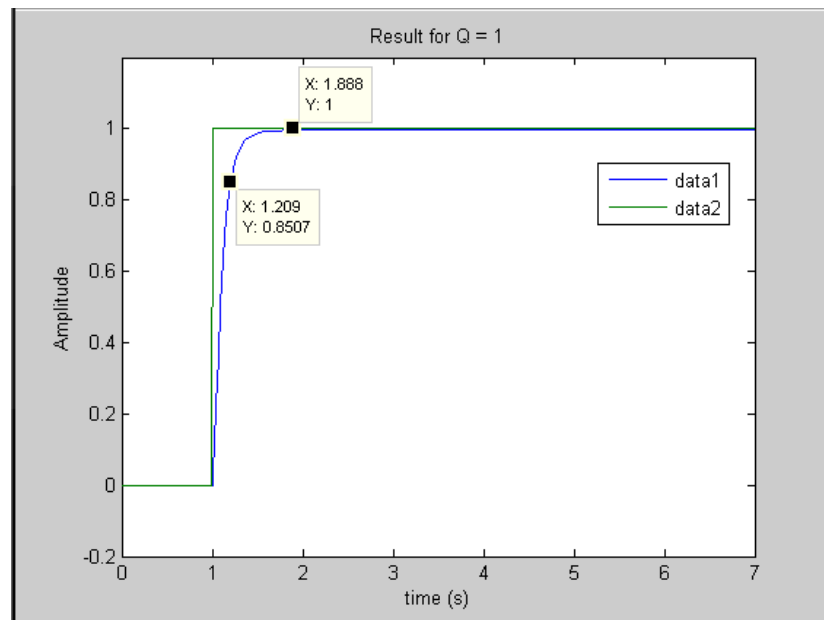


Figure 4.7: Result for Q = 1

The graph shows that the servo motor achieves its maximum speed at 0.8507 rad/s and it takes 0.888 seconds to reach its steady-state.

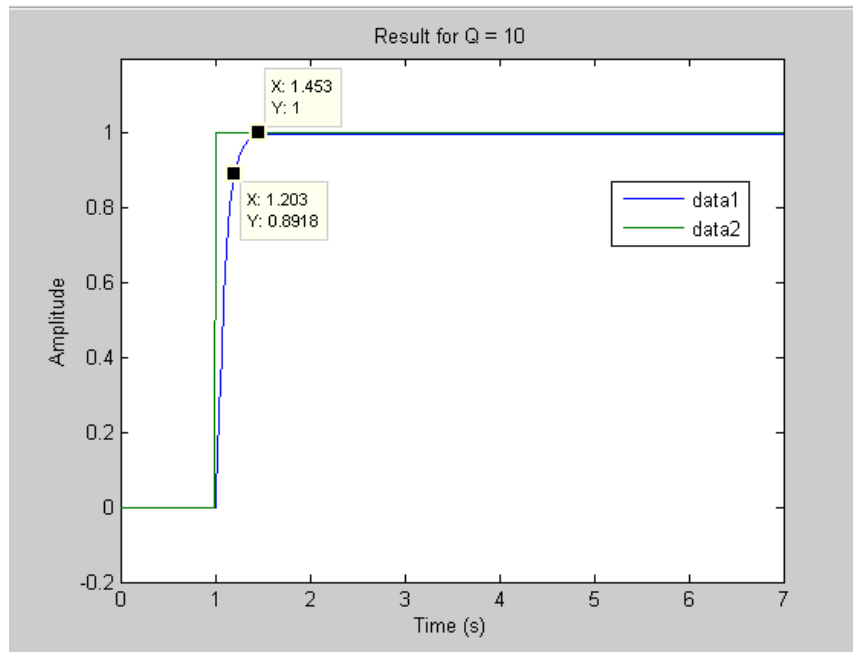


Figure 4.8: Result for Q = 10

Figure 4.8 shows the servo motor achieves its maximum speed at 0.8918 rad/s and it takes 0.453 seconds to reach its steady-state.

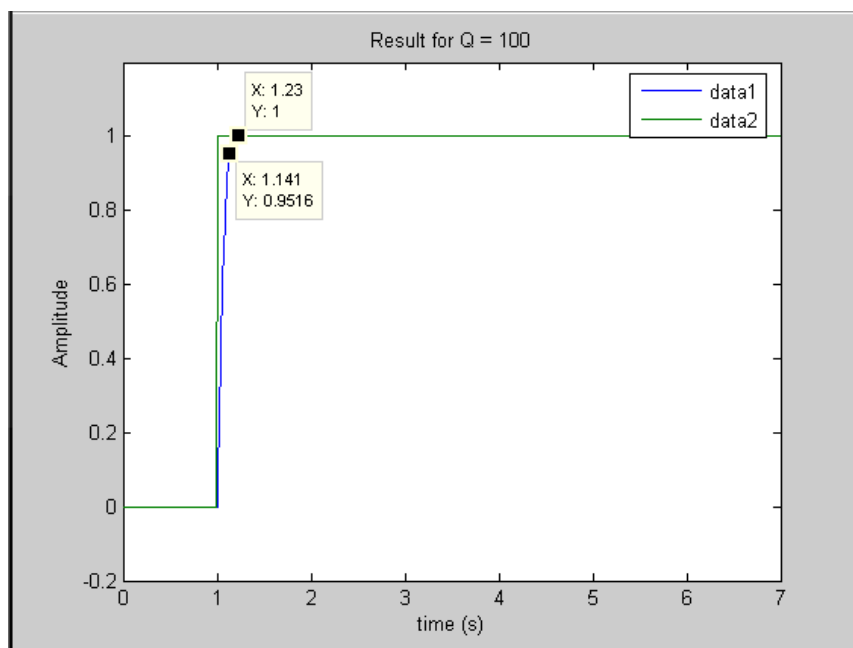


Figure 4.9: Result for Q = 100

Figure 4.9 shows that the motor achieve its maximum speed at 0.9516 rad/s and it takes 0.23 seconds to reach its steady-state.

The results from the simulation of the three different values of  $Q$  are shown in Table 4.2.

Table 4.2: Simulation results for three different  $Q$

$Q$	$R$	$K$ (Gain)	Time Rise	Settling Time
$[1 \ 0 ; 0 \ 1]$	1	$K = [0.2450 \ 0.0801]$	0.209	0.888
$[1 \ 0 ; 0 \ 10]$	1	$K = [0.3241 \ 0.8025]$	0.203	0.453
$[1 \ 0 ; 0 \ 100]$	1	$K = [0.7704 \ 5.3505]$	0.141	0.23

From the table above, the time for the system to rise and settled decreased due to the increased of the value  $Q$ .

## **CHAPTER 5**

### **CONCLUSION AND RECOMMENDATION**

#### **5.1 Introduction**

This thesis has discussed on development of Matlab GUI and Linear Quadratic Regulator which has ability to control the speed of the motor. This project is implemented using G340 motor driver and interfacing between servomotor and computer through serial and parallel port with a development program using Matlab GUI.

This project has been able to achieve some of its objectives. Some of the objectives that did not achieve will be discussed about its problem in this chapter. Then, conclusion will be made to state out the strength and weakness of the project. Lastly, planning for future work will be discussed.

#### **5.2 Assessment of design**

While developing the application design, several difficulties and mistakes were countered. This project required some research and troubleshooting at the motor driver. During the project development, several problems are faced. They are:

- The servo motor control

Unfamiliar with servo motor has caused the slow down on selecting either pulse or voltage need to send to the servo motor to control the speed.

- Motor driver application

Don't know how to connect the motor driver to the serial and parallel port has taking much time. The motor driver also had its own controller that is PID controller and makes this project difficult because this project develops its own controller that is Linear Quadratic Regulator.

- Graphical user interface

There need a time to learn the function of in the Matlab GUI for built the user interface and for sending data through serial and parallel port.

### 5.3 Strength and Weakness

This project offer simple and efficient way to construct a GUI to control the speed of the servo motor. Most of the controlling speed use such complex mathematical equation and advance control system to control the speed.

The user friendly interface in computer can let the user easily control the speed of the motor manually. By changing the value in the text box then the servo motor will change its speed according to the value insert before.

The weakness of this project is, it can't specify the speed of the motor. There is no display for the speed of the motor either in revolution per minute (rpm) or other unit. So that we don't know the speed of the servo motor is running.

## 5.4 Suggestion for Future Work

The project is functioning well from the user friendly interface and its simulation. However, in implementing it, need to be improved to a more advanced and better application in the future. For future improvement, several suggestions are proposed:

- Suitable motor driver  
The motor driver that used in this project need to be no its own controller, so that this project can use its own controller that has been develop.
- Choose familiar servo motor  
Unfamiliar with the servo motor makes this project hard because the servo motor is the main part.

## 5.5 Costing & Commercialization

There is no cost at all to develop this project because all the instrument and hardware part is provided by the FKEE Laboratory. This project is generally can be implemented more on subject at the university due to its cost and this project itself.

This is said that because this project is focus on the LQR controller where this controller is still on research nowadays. It is new knowledge so that it is good if implemented on subject at the university.

Commercialization for this project can be done among all universities or others higher study institutes relate to the technical skills. It can be very useful technical learning tools for the engineering laboratory that are related to Control Systems topic.

## REFERENCES

- [1] F. L. Lewis (1999). "EE 4343/5329 - Control System Design Project".
- [2] Norman S. Nise (2004). "Control System Engineering". John Wiley & Sons, Inc.
- [3] Sergey E. Lyshevski. "Electromechanical Systems, Electric Mechanics AND Applied Mechatronics"
- [4] Wikipedia (2008), Data acquisition  
URL [http://en.wikipedia.org/wiki/Data\\_acquisition](http://en.wikipedia.org/wiki/Data_acquisition)
- [5] Hello and welcome to Mik's page at DSP lab.  
URL <http://dsp.ucsd.edu/students/present-students/mik/matlabgui/>
- [6] Wikipedia (2008), Linear Quadratic Regulator  
URL [http://en.wikipedia.org/wiki/Linear-quadratic\\_regulator](http://en.wikipedia.org/wiki/Linear-quadratic_regulator)
- [7] Wikipedia (2008), Servo Motor  
URL [http://en.wikipedia.org/wiki/Servo\\_motor](http://en.wikipedia.org/wiki/Servo_motor)

## **G340 INSTALLATION NOTES**

**(March 31, 2002)**

Thank you for purchasing the G340 drive. The G340 DC servodrive is warranted to be free of manufacturing defects for 1 year from the date of purchase. Also anyone who is dissatisfied with it or is unable to make it work will be cheerfully refunded the purchase price if the G340 is returned within 15 days of the purchase date.

### **PLEASE READ FIRST BEFORE USING THE G340:**

If you are not familiar with DC servodrives please do the following setup instructions with the motor on the bench before mounting it on the mechanism it will eventually run. This will allow you to get a baseline motor behavior of what to expect.

Before you start, you must have a suitable encoder mounted and properly aligned on the motor. Follow the manufacturer's instructions on mounting and aligning the encoder if the motor doesn't already come with one.

Next you must have a DC power supply suitable for the motor. Do not use a power supply voltage more than 5 volts in excess of the motors rated voltage. The power supply current rating must equal the maximum current you expect to run the motor at.

Finally have a STEP and DIRECTION pulse source available.

Before going on, turn the current LIMIT trimpot a quarter to half of full scale. Turn the GAIN trimpot fully off and turn the DAMPING trimpot to a quarter of full scale. The trimpots are single-turn, do not over-torque them with a screwdriver.

### **REMOVING AND REPLACING THE COVER:**

The STEP PULSE MULTIPLIER and the INPUT OPTION settings are jumpered internally. It is necessary to remove the cover of the drive to change these settings from their default values. Please follow the steps below on how to remove and replace the cover to avoid damaging the drive:

#### **REMOVING THE COVER:**

- 1) Remove the two 2-56 phillips-head screws on the bottom of the drive.
- 2) Gently lift the back of the cover upward until the LED clears the rectangular hole
- 3) Slide the cover backwards while still lifting until it clears the drive.

#### **REPLACING THE COVER:**

- 1) Make sure the LED is vertical relative to the drive printed circuit board.
- 2) Slide the cover forward over the drive while lifting the back of the cover.
- 3) When the cover is fully forward, look to see the LED is aligned with the hole.
- 4) Gently press the cover down, making sure the LED moves into the hole.
- 5) Replace the screws on the bottom of the drive.

It is recommended to use small needle-nose pliers or tweezers to move the jumpers on the internal headers.

### **G340 MULTIPLIER AND INPUT OPTION HEADERS:**

#### **MULTIPLIER OPTION HEADER**

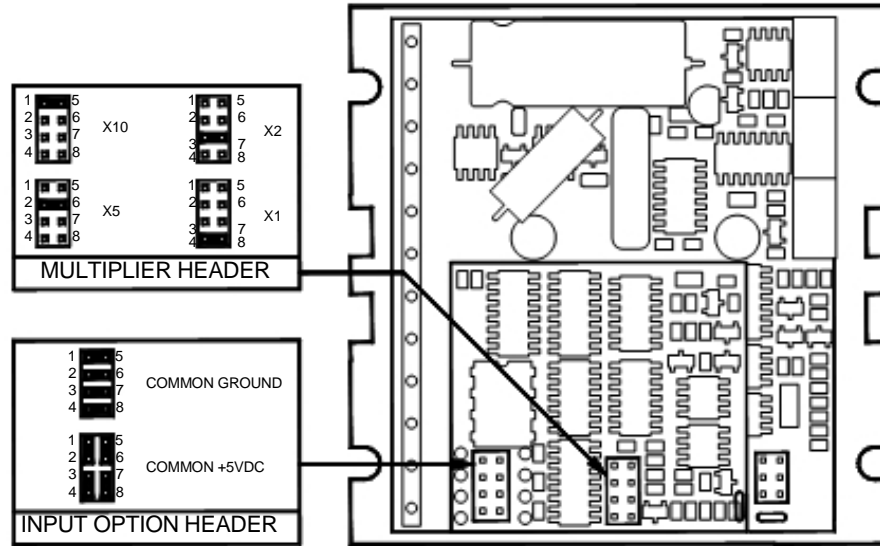
The G340 has a built in STEP PULSE MULTIPLIER. This circuit puts out 1, 2, 5, or 10 pulses for every input STEP pulse. All motor moves will be in these increments. For example, if X10 is selected, then the motor will move 10 encoder counts for every STEP pulse sent to the G340. Move the jumper on the MULTIPLIER HEADER to select the desired multiplier. Do not operate the drive without a jumper.

#### **INPUT OPTION HEADER**

The INPUT OPTION HEADER allows the STEP and DIRECTION opto-isolators to be configured as either common ground or common +5VDC.

If the G340 inputs are driven by a source that has only ground available, such as a PC parallel port, move the 4 jumpers on the header so that it looks like the COMMON GROUND setting. Connect the input driver ground to TERM 12 on the main connector.

If the G340 inputs are driven by open collector transistors or standard TTL gates, move the 4 jumpers on the header so that it looks like the COMMON +5VDC setting.



#### **G340 TERMINAL WIRING:**

**IMPORTANT:** When first testing the G320, connect ERR/RES (term. 5) to ENC+ (term. 7). Please follow the next steps in the sequence they are given.

#### **STEP 1: ENCODER HOOK-UP**

The encoder must be at minimum a 25 line-count digital quadrature encoder and must operate on a single +5VDC power supply. If the encoder supply current is more than 50 mA, use an external +5VDC supply. It may have an INDEX output, which will not be used. If it has differential outputs, use only the "+" phase outputs. **IMPORTANT: Connect a 470-ohm resistor from TERM. 6 to TERM. 7 if an external power supply is used for the encoder.**

**(TERM. 6) ENC-** Connect the encoder power supply ground to this terminal.

**(TERM. 7) ENC+** Connect the encoder +5VDC to this terminal

**(TERM. 8) PHASE A** Connect the encoder phase "A" to this terminal

**(TERM. 9) PHASE B** Connect the encoder phase "B" to this terminal

#### **STEP 2: POWER SUPPLY HOOK-UP**

Keep the power supply leads short and use the largest wire gauge that will fit in the terminals. If the lead length is more than 18" use a 1000 uF capacitor across the G340 power supply terminals. Make sure your power supply can provide the peak current the motor may draw. The power supply voltage must be between 18 VDC and 80 VDC. The actual voltage should not be more than 5 volts higher than the motor's rated voltage.

**(TERM. 1) POWER GROUND** Connect the power supply ground to this terminal.

**(TERM. 2) +18 TO 80 VDC** Connect the power supply "+" to this terminal

#### **STEP 3: TESTING THE ENCODER**

At this point the encoder should be tested for functionality. If you wish to monitor the POSITION ERROR test point with a voltmeter or oscilloscope, then remove the cover of the drive now. If you have a choice, pick the oscilloscope.

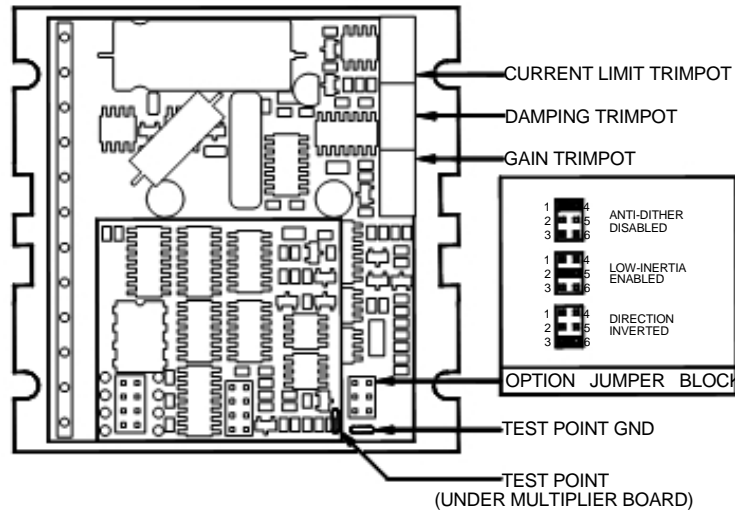
The POSITION ERROR test point shows the difference between the command position and the actual motor position. When both are the same, the voltage will be +5VDC. For every count the motor is clockwise of the command position, the voltage will decrease by 0.04 volts. When it drops to 0.4 volts, the protection circuit takes over and resets the drive for 3 seconds. While reset, the FAULT light is on.

For counter-clockwise position errors the voltage will increase by 0.04 volts for every count until it reaches 9.6 volts when again the protection circuit takes over as before.

**VOLTMETER MONITORING:** Place the red lead on the test point and the black lead on the large blue capacitor lead (GND) furthest from the main connector. Turn on the power supply. The FAULT light should be on for 3 seconds and then

turn off. The voltmeter should read +5VDC. Turn the motor shaft clockwise VERY slowly. The voltmeter reading should decrease 0.04 volts for every encoder count. When the reading reaches 0.4 volts, the red light will turn on and the voltage will jump back to +5VDC. After 3 seconds the light will turn off. You may turn the motor shaft counter-clockwise as well. The voltage will increase then by 0.04 volts per count until it reaches 9.6 volts and trips the protection circuit.

**OSCILLOSCOPE MONITORING:** Set the scope to 2 volts / cm vertical and about 1 millisec per cm horizontal. Zero the trace to the bottom line on the screen. DC couple the input. Place the probe on the test point and the ground clip to the blue capacitor ground lead. Follow the steps in **VOLTMETER TESTING** above.



#### **STEP 4: CONTROL INPUT HOOK-UP**

The control input group is the standard step motor drive STEP, DIRECTION and +5VDC lines. The STEP and DIRECTION signal drivers must be TTL compatible and have edge transition times of 100 ns or faster. The +5VDC is the opto-isolator common anode line and must be returned to the pulse source +5VDC supply.

**(TERM. 10) DIR** Connect the DIRECTION line to this terminal.

**(TERM. 11) STEP** Connect the STEP line to this terminal.

**(TERM. 12) +5VDC** Connect this terminal to the controller +5VDC power supply

#### **STEP 5: TESTING THE CONTROL INPUTS**

You may wish to test the functionality of these inputs. If you used an oscilloscope in the previous section, leave it connected to the test point. If you used a voltmeter, then remove it from the drive.

Set the STEP pulse generator to about 40 pulses per second and set the DIRECTION output to clockwise (logical "1"). Turn on the power supply. After the power-on reset period of 5 seconds the FAULT light will turn off.

If you are using an oscilloscope, then the test point voltage will begin to increase until 3 seconds later it trips the protection circuit at 9.6 volts. The FAULT light will turn on for 5 seconds and voltage will snap back to +5VDC. After the FAULT turns off, the sequence will repeat again.

If you are not using an oscilloscope, just see if the FAULT light turns on and off every three seconds.

#### **STEP 6: MOTOR HOOK-UP**

Make sure the power is off and the STEP pulse source is set to zero pulses per second. Check to see if the trimpot settings are set according to the instructions on page 2. You may wish to secure the motor so it can't jump off the bench.

**(TERM. 3) ARM-** Connect the RED motor lead to this terminal.

**(TERM. 4) ARM+** Connect the BLACK motor lead to this terminal.

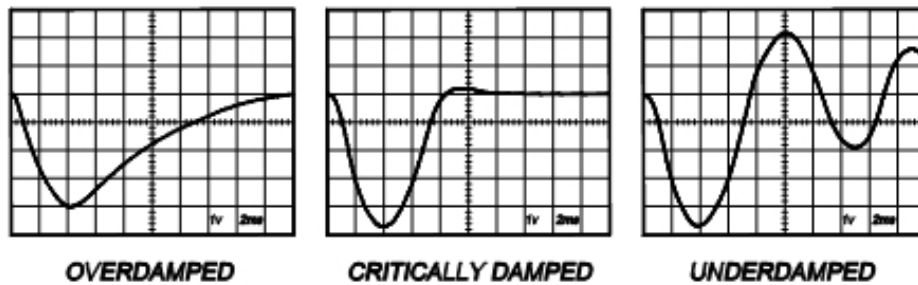
#### **STEP 7: TUNING THE SERVO**

Turn on the power supply. The FAULT light should turn off after 3 seconds. If everything is correct you should hear the motor "singing". This is normal. The motor is dithering or bouncing between adjacent encoder counts. The integral term in a PID loop has infinite DC gain over time and will amplify even the smallest position error. Because encoder feedback can only occur on count edges, the loop is "blind" until it encounters an encoder count edge. It then reverses the motor direction until another edge is found, then the process repeats.

If the motor jumps slightly and the FAULT light immediately turns back on, then either the motor is wired backwards or the trim pots are misadjusted. Check the trim pot settings. If they seem right then switch the motor leads and try again. If it still doesn't work and you followed all the previous steps, call me at the number at the end of this document.

Now turn on your STEP pulse source and ramp the speed up to see if the motor turns. It should turn clockwise with a logical "1" on the DIRECTION input.

The optimum way to tune the servo is to induce an impulse load on the motor while watching an oscilloscope to see how the motor behaves in response, then adjusting the PID co-efficients (**GAIN** and **DAMPING** trim pot settings) for optimal behavior.



In all cases the motor must return to the command position, what matters is how it does it. The manner in which the motor returns to its command position is called damping. At one extreme called overdamped response, the motor returns to position after a long, drawn out delay. At the other extreme called underdamped response, the motor returns to its position too rapidly, overshoots, returns and undershoots and so on until it finally settles at its command position. This is also caused by ringing; when extreme, the over/undershoot builds in amplitude until the motor enters violent oscillation. Between the two extremes is the optimal response called critical damping. Here the motor rapidly returns to its position with little or no overshoot in the minimal amount of time.

#### **GAIN AND DAMPING**

GAIN and DAMPING settings generally track each other. If you increase GAIN (greater stiffness), then increased DAMPING is needed as well to restore critical damping. Be careful, increasing GAIN without increasing DAMPING may cause the motor to break out into violent oscillation.

The higher GAIN is set, the noisier the motor will be when stopped. This is because higher gain causes more vigorous dithering between encoder counts at rest. There is a trade-off between high gain (high stiffness) on the one hand and excessive dithering (noise and motor heating) on the other. Use judgment here.

To see how your servo is compensated it is first necessary to induce a disturbance. The easiest way is to switch the DIRECTION input while commanding a constant speed via the STEP input. The abrupt direction change puts just the momentary load needed on the motor while you watch how it responds.

If you are using an oscilloscope, use channel 1 on the test point and channel 2 on the DIRECTION input. Set the trigger to "normal", trigger source to channel 2 and trigger edge to "+". You should see a single sweep for every clockwise change in direction.

Slowly increase STEP speed until you get a picture similar to one of the three above, and then do the following:

- 1) OVERDAMPED: Decrease DAMPING or increase GAIN
- 2) CRITICALLY DAMPED: Do nothing; you're there
- 3) UNDERDAMPED: Decrease GAIN or increase DAMPING

#### **(POSITION ERROR TEST POINT NOTE)**

Don't confuse the POSITION ERROR with the motor or machine position. The signal is actually the differential position error between the command speed and the motor speed. As noted above, sending clockwise STEP pulses moves the POSITION ERROR voltage more positive while turning the motor clockwise moves the POSITION ERROR voltage more negative.

When the motor encoder counts match the number of STEP pulses being sent one for one, the POSITION ERROR voltage stays at +5VDC. If the motor gets ahead of the STEP pulses such as during very rapid deceleration, the voltage will decrease by 0.04 volts for every encoder count the motor is ahead of the STEP pulses sent. The PID algorithm will force the motor to match the STEP input over time and restore the POSITION ERROR voltage back to +5 VDC.

## CURRENT LIMIT

The current LIMIT trimpot sets maximum current the motor is permitted to have. It is adjustable from 0 amps to 20 amps. Normally the LIMIT trimpot is set to maximum (20 amps) unless you want to limit motor torque to a lower value.

Motor speed and position is unaffected by the current LIMIT setting unless the torque demand due to load exceeds this setting, then the motor position will fall behind the command position because of insufficient torque.

## FAULT INDICATOR

The FAULT indicator is on while the drive is in power-on reset, the DISABLE input is held "low" or if the protection circuit is tripped due to a fault condition. All power MOSFETs are turned off and all internal counters are reset. The FAULT condition lasts for 3 seconds, and then self-resets to try again. If the protection circuit tripped it and the cause is not cleared, then it will immediately re-enter the FAULT state again and repeat the cycle.

There are two conditions that will trip the protection circuit. One condition is if a short-circuit occurs and current exceeds 20 amps. The other condition is if the POSITION ERROR exceeds +/- 120 counts causing a break of the servo-lock. This condition can have several causes:

- 1) The loop settings are severely under-damped and the motor breaks out into oscillation.
- 2) Excessive motor load due to acceleration or workload.
- 3) The speed command in excess of what the motor can deliver.
- 4) The current LIMIT is set too low.
- 5) The power supply current is insufficient for the demand.
- 6) The power supply voltage is below 18 VDC.
- 7) The motor is wired backwards, is broken or disconnected.
- 8) Encoder failure.

## REVERSING DEFAULT MOTOR DIRECTION

The G340 will turn the motor in the CW direction when the DIRECTION input is "high" (logical "1", or +5VDC). If instead CCW is preferred, then:

- 1) Reverse the motor "+" and "-" leads (term. 3 with term. 4)
- 2) Reverse the encoder "channel A" and "channel B" leads (term. 8 with term. 9)

## USING THE G340 WITH VERY SMALL MOTORS

Very small motors have low inductance relative to their operating current. Consequently ripple current due to pulse-width modulation can quickly overheat and destroy these motors. If the G340 will be used with these motors, then an external low pass filter must be used to attenuate ripple current to tolerable levels.

A suggested filter consists of two 150 microhenry inductors in series with each motor lead and a 2 microfarad, low inductance film capacitor across the motor leads. The inductors must be rated for the anticipated peak motor current.

This filter is also needed if ironless-armature or "pancake" motors are used. These motors have very low inductance as well and will overheat if driven directly by the G340.

## (TERM. 5) ERR / RES

This terminal functions as an ERROR output and as a RESET input. Because this terminal functions as both an input and an output, some detailed description is necessary.

When first testing the G340, ERR/RES (term. 5) was connected to ENC+ (term. 7). It can be left that way if it is not necessary to read the state of the ERROR output. Otherwise, the following details are important.

The ERROR output is latched in the "ERROR" state (term. 5 = "0") by the power-on reset circuitry in the G340. It will stay in this state indefinitely until it is cleared by applying +5V to this terminal for at least 1 second.

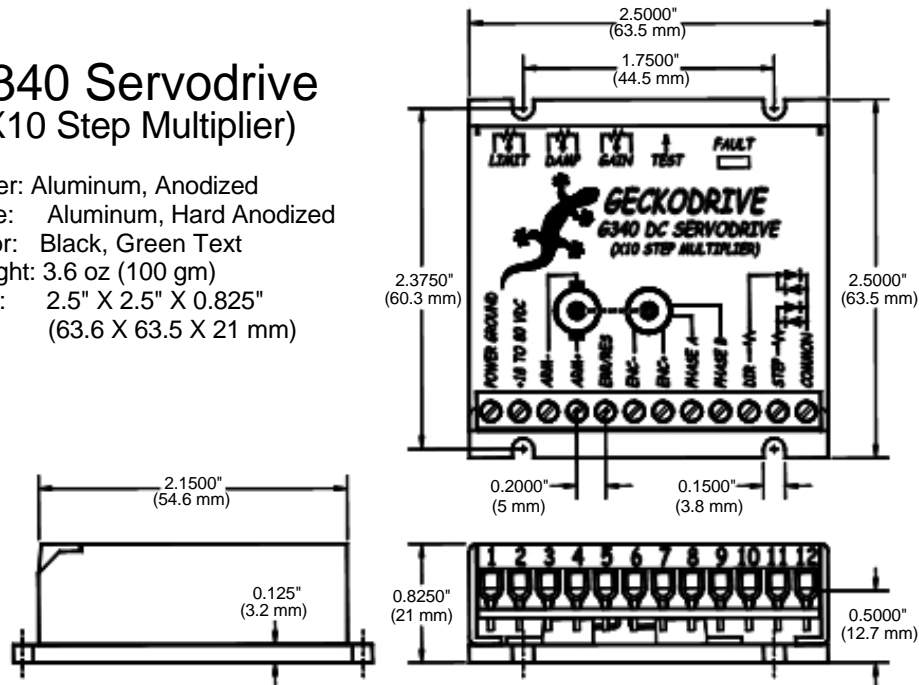
The voltage on this terminal is +5VDC when the G340 is functioning normally. The voltage on this terminal goes to 0VDC whenever the FAULT indicator is lit. This output can be used to signal your controller that an error has occurred.

Normally when the G340 is first powered up, it will be necessary to push the momentary switch to START for 5 seconds. This will clear the power-on reset condition and extinguish the FAULT LED. The motor will then be enabled and the drive will begin to operate. If at anytime after that a condition occurs that causes the G340 to "fault out", such as not being able to complete a step command, the ERR/RES terminal will go to "0", signaling the computer an error has occurred. This will require the operator to correct the problem that caused the fault and then push the switch to "START" for 5 seconds to re-enable the G340.



## G340 Servodrive (X10 Step Multiplier)

Cover: Aluminum, Anodized  
 Plate: Aluminum, Hard Anodized  
 Color: Black, Green Text  
 Weight: 3.6 oz (100 gm)  
 Size: 2.5" X 2.5" X 0.825"  
 (63.6 X 63.5 X 21 mm)



### G340 SPECIFICATIONS:

Power Supply	18 to 80 VDC
Motor Current	0 to 20 Amps
Lock Range	+/- 128 count following error
Feedback	Quadrature TTL Encoder
Feedback Resolution	X4 Encoder Line Count
Switching Frequency	25 kHz
Current Limit	0 to 20 Amp, Trimpot Adjustable
Analog PID	Damping and Gain Trimpots
Step Pulse Frequency	0 to 250 kHz
Step Pulse "0" Time	0.5 Microseconds Min.
Step Pulse "1" Time	3.5 Microseconds Min.
Multiplier Settings	X1, X2, X4, X5 and X10
Size	2.5" X 2.5" X 0.825"
Weight	3.6 oz weight
Encoder Supply	+5VDC, 50 mA max

Geckodrive Inc.  
 9702 Rangeview Drive  
 Santa Ana, CA 92705

Phone: 1-714-771-1662  
 Fax: 1-714-771-4867  
 Web Site: [www.geckodrive.com](http://www.geckodrive.com)

## **Appendix B**

### **Coding Program**

```

function varargout = FIND_K(varargin)
% FIND_K M-file for FIND_K.fig
%   FIND_K, by itself, creates a new FIND_K or raises the existing
%   singleton*.
%
%   H = FIND_K returns the handle to a new FIND_K or the handle to
%   the existing singleton*.
%
%   FIND_K('CALLBACK',hObject,eventData,handles,...) calls the local
%   function named CALLBACK in FIND_K.M with the given input arguments.
%
%   FIND_K('Property','Value',...) creates a new FIND_K or raises the
%   existing singleton*. Starting from the left, property value pairs are
%   applied to the GUI before FIND_K_OpeningFunction gets called. An
%   unrecognized property name or invalid value makes property application
%   stop. All inputs are passed to FIND_K_OpeningFcn via varargin.
%
%   *See GUI Options on GUIDE's Tools menu. Choose "GUI allows only one
%   instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES
%
% Copyright 2002-2003 The MathWorks, Inc.
%
% Edit the above text to modify the response to help FIND_K
%
% Last Modified by GUIDE v2.5 15-Oct-2008 00:11:31
%
% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',    mfilename, ...
                  'gui_Singleton', gui_Singleton, ...
                  'gui_OpeningFcn', @FIND_K_OpeningFcn, ...
                  'gui_OutputFcn', @FIND_K_OutputFcn, ...
                  'gui_LayoutFcn', [] , ...
                  'gui_Callback', []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});

```

```

end

if nargin
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end

% End initialization code - DO NOT EDIT

% --- Executes just before FIND_K is made visible.
function FIND_K_OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles     structure with handles and user data (see GUIDATA)
% varargin    command line arguments to FIND_K (see VARARGIN)

% Choose default command line output for FIND_K
handles.output = hObject;

% Update handles structure
guidata(hObject, handles);

% UIWAIT makes FIND_K wait for user response (see UIRESUME)
% uiwait(handles.figure1);

% --- Outputs from this function are returned to the command line.
function varargout = FIND_K_OutputFcn(hObject, eventdata, handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles     structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure

```

```
varargout{1} = handles.output;
```

```
function Ra_Callback(hObject, eventdata, handles)
% hObject    handle to Ra (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of Ra as text
%        str2double(get(hObject,'String')) returns contents of Ra as a double
data1 = str2double(get(hObject, 'String'));
if isnan(data1)
    set(hObject, 'String', '');
    errordlg('    Input Must Be A Number !!','Error');
end
```

```
% --- Executes during object creation, after setting all properties.
function Ra_CreateFcn(hObject, eventdata, handles)
% hObject    handle to Ra (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%        See ISPC and COMPUTER.
if ispc
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end
```

```
function La_Callback(hObject, eventdata, handles)
% hObject    handle to La (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
```

```

% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of La as text
%    str2double(get(hObject,'String')) returns contents of La as a double
data1 = str2double(get(hObject, 'String'));
if isnan(data1)
    set(hObject, 'String', '');
    errordlg('    Input Must Be A Number !!','Error');
end

% --- Executes during object creation, after setting all properties.
function La_CreateFcn(hObject, eventdata, handles)
% hObject    handle to La (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%    See ISPC and COMPUTER.
if ispc
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end

function Bm_Callback(hObject, eventdata, handles)
% hObject    handle to Bm (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of Bm as text
%    str2double(get(hObject,'String')) returns contents of Bm as a double
data1 = str2double(get(hObject, 'String'));
if isnan(data1)

```

```

    set(hObject, 'String', '');
    errordlg('    Input Must Be A Number !!','Error');
end

% --- Executes during object creation, after setting all properties.
function Bm_CreateFcn(hObject, eventdata, handles)
% hObject    handle to Bm (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end

function Kemf_Callback(hObject, eventdata, handles)
% hObject    handle to Kemf (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of Kemf as text
%       str2double(get(hObject,'String')) returns contents of Kemf as a double
data1 = str2double(get(hObject, 'String'));
if isnan(data1)
    set(hObject, 'String', '');
    errordlg('    Input Must Be A Number !!','Error');
end

% --- Executes during object creation, after setting all properties.
function Kemf_CreateFcn(hObject, eventdata, handles)

```

```

% hObject    handle to Kmf (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%         See ISPC and COMPUTER.
if ispc
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end

```

```

function Kt_Callback(hObject, eventdata, handles)
% hObject    handle to Kt (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of Kt as text
%         str2double(get(hObject,'String')) returns contents of Kt as a double
data1 = str2double(get(hObject, 'String'));
if isnan(data1)
    set(hObject, 'String', "");
    errordlg('    Input Must Be A Number !!','Error');
end

```

```

% --- Executes during object creation, after setting all properties.
function Kt_CreateFcn(hObject, eventdata, handles)
% hObject    handle to Kt (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%         See ISPC and COMPUTER.

```

```

if ispc
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end

```

```

function J_Callback(hObject, eventdata, handles)
% hObject    handle to J (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of J as text
%        str2double(get(hObject,'String')) returns contents of J as a double
data1 = str2double(get(hObject,'String'));
if isnan(data1)
    set(hObject,'String','');
    errordlg('    Input Must Be A Number !!','Error');
end

```

```

% --- Executes during object creation, after setting all properties.
function J_CreateFcn(hObject, eventdata, handles)
% hObject    handle to J (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%        See ISPC and COMPUTER.
if ispc
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end

```

```

% --- Executes on button press in run1.
function run1_Callback(hObject, eventdata, handles)

Ra1 = str2double(get(handles.Ra,'String'));
La1 = str2double(get(handles.La,'String'));
Bm1 = str2double(get(handles.Bm,'String'));
Kemf1 = str2double(get(handles.Kemf,'String'));
Kt1 = str2double(get(handles.Kt,'String'));
J1 = str2double(get(handles.J,'String'));

A=[-Ra1/La1 -Kemf1/La1; Kt1/J1 -Bm1/J1];
B=[1/La1;0];
C=[0 1;1 0];
D=[0;0];

Q=[1 0;0 1];
R=[1];

[K]=lqr(A,B,Q,R);
set(handles.answer,'String',K(1, 1));
set(handles.answer2,'String',K(1, 2));

% hObject    handle to run1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% --- Executes during object creation, after setting all properties.
function text1_CreateFcn(hObject, eventdata, handles)
% hObject    handle to text1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

```

```

function answer_Callback(hObject, eventdata, handles)
% hObject    handle to answer (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of answer as text
%        str2double(get(hObject,'String')) returns contents of answer as a double

```

```

% --- Executes during object creation, after setting all properties.
function answer_CreateFcn(hObject, eventdata, handles)
% hObject    handle to answer (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

```

```

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end

```

```

function Ra1_Callback(hObject, eventdata, handles)
% hObject    handle to Ra (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of Ra as text
%        str2double(get(hObject,'String')) returns contents of Ra as a double

```

```

% --- Executes during object creation, after setting all properties.
function Ra1_CreateFcn(hObject, eventdata, handles)
% hObject    handle to Ra (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end

function Kemf1_Callback(hObject, eventdata, handles)
% hObject    handle to Kemf (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of Kemf as text
%       str2double(get(hObject,'String')) returns contents of Kemf as a double
data1 = str2double(get(hObject, 'String'));
if isnan(data1)
    set(hObject, 'String', '');
    errordlg('    Input Must Be A Number !!','Error');
end

% --- Executes during object creation, after setting all properties.
function Kemf1_CreateFcn(hObject, eventdata, handles)
% hObject    handle to Kemf (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB

```

```

% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%    See ISPC and COMPUTER.
if ispc
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end

% --- Executes during object creation, after setting all properties.
function axes2_CreateFcn(hObject, eventdata, handles)
% hObject    handle to axes2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: place code in OpeningFcn to populate axes2
[c,map]=imread('motor','JPEG');
image(c)
set(gca,'visible','off')

function answer2_Callback(hObject, eventdata, handles)
% hObject    handle to answer2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of answer2 as text
%    str2double(get(hObject,'String')) returns contents of answer2 as a double

% --- Executes during object creation, after setting all properties.
function answer2_CreateFcn(hObject, eventdata, handles)

```

```

% hObject    handle to answer2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%         See ISPC and COMPUTER.
if ispc
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end

```

```

% --- Executes on button press in clr.
function clr_Callback(hObject, eventdata, handles)
% hObject    handle to clr (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
set(handles.La,'string','');
set(handles.Ra,'string','');
set(handles.Bm,'string','');
set(handles.Kt,'string','');
set(handles.Kemf,'string','');
set(handles.J,'string','');
set(handles.answer,'string','');
set(handles.answer2,'string','');

```

```

% --- Executes on button press in main1.
function main1_Callback(hObject, eventdata, handles)
% hObject    handle to main1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
close

```

```
figure(main)

% --- Executes on button press in next.
function next_Callback(hObject, eventdata, handles)
% hObject    handle to next (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
close
open('H:\PSm\PSM2.mdl')
```