DESIGN OF AN INTEGRATED HARDWARE-IN-THE-LOOP SIMULATION SYSTEM

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES
OF
MIDDLE EAST TECHNICAL UNIVERSITY

BY

SERDAR ÜŞENMEZ

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR
THE DEGREE OF MASTER OF SCIENCE
IN
MECHANICAL ENGINEERING

JUNE 2010

Approval of the thesis:

DESIGN OF AN INTEGRATED HARDWARE-IN-THE-LOOP SIMULATION SYSTEM

submitted by **SERDAR ÜŞENMEZ** in partial fulfillment of the requirements for the degree of **Master of Science in Mechanical Engineering Department, Middle East Technical University** by,

Date: 22/06/2010

ABSTRACT

DESIGN OF AN INTEGRATED HARDWARE-IN-THE-LOOP SIMULATION SYSTEM

Üşenmez, Serdar
M.Sc., Department of Mechanical Engineering
Supervisor: Assist. Prof. Dr. A. Buğra Koku
Co-Supervisor: Assist. Prof. Dr. Melik Dölen

June 2010, 172 Pages

This thesis aims to propose multiple methods for performing a hardware-in-the-loop simulation, providing the hardware and software tools necessary for design and execution. For this purpose, methods of modeling commonly encountered dynamical system components are explored and techniques suitable for calculating the states of the modeled system are presented. Modules and subsystems that enable the realization of a hardware-in-the-loop simulation application and its interfacing with external controller hardware are explained. The thesis also presents three different simulation scenarios. Solutions suitable for these scenarios are provided along with their implementations. The details and specifications of the developed software packages and hardware platforms are given. The provided results illustrate the advantages and disadvantages of the approaches used in these solutions.

Keywords: Hardware in the Loop Simulation, Dynamic System Modeling, Control Systems Education, Peripheral Device Emulation

ÖZ

TÜMLEŞİK BİR ÇEVRİMİÇİ DONANIM BENZETİMİ SİSTEMİNİN TASARLANMASI

Üşenmez, Serdar
Yüksek Lisans, Makine Mühendisliği Bölümü
Tez yöneticisi: Yrd. Doç. Dr. A. Buğra Koku
Yardımcı tez yöneticisi: Yrd. Doç. Dr. Melik Dölen

Haziran 2010, 172 Sayfa

Bu tezde bir çevrimiçi donanım benzetiminin gerçekleştirilmesi için çeşitli yöntemlerin öne sürülmesi ve tasarım ile uygulama için gerekli donanım ve yazılım gereçlerinin sağlanması hedeflenmiştir. Bu amaçla, sıklıkla karşılaşılan dinamik sistem bileşenlerinin modellenmesine dair yöntemler incelenmiş ve modellenen sisteme ait durum değişkenlerinin hesaplanmasına yönelik yordamlar sunulmuştur. Bir çevrimiçi donanım benzetimi uygulamasının gerçekleştirilmesini ve bunun harici denetleyici donanımlarla arabağlanmasını sağlayan modüller ve alt sistemler açıklanmıştır. Bu tez ayrıca üç farklı benzetim senaryosu sunmaktadır. Bu senaryolara uygun çözümler ve bunların uygulanışları anlatılmıştır. Geliştirilen yazılım paketleri ve donanım platformlarına dair ayrıntılar ve özellikler verilmiştir. Sunulan sonuçlar, bu çözümlerde kullanılan yaklaşımların faydalı ve kusurlu yönlerini gösterir niteliktedir.

Anahtar kelimeler: Çevrimiçi Donanım Benzetimi, Dinamik Sistem Modellemesi, Kontrol Sistemleri Eğitimi, Çevresel Cihaz Öykünmesi

To everyone I care,

“I don't know half of you half as well as I should like; and I like less than half of you half as well as you deserve.”

– Bilbo Baggins, The Lord of the Rings

ACKNOWLEDGEMENTS

First of all, I express my sincere appreciation to Asst. Prof. Dr. Ahmet Buğra Koku and Asst. Prof. Dr. Melik Dölen for generously providing their knowledge, wisdom, guidance and support during my studies regarding this thesis.

I also thank sincerely my colleagues Barış Ragıp Mutlu, Ulaş Yaman, Ergin Kılıç and Rasim Aşkın Dilan for their endless support and collaboration during all our works. I also thank all my friends, whose support has given me and my colleagues the will to work.

I should give special thanks to all the students who took the ME 534 – Computer Control of Machines course given in Middle East Technical University during Spring 2008 and Fall 2010 semesters for bearing with me throughout their already-painful final project period. Without their precious feedback, this thesis would be missing much, much more than just a section.

I would like to express my thanks and love for my parents, for all their love, patience and understanding, with hopes of one day proving myself worthy of their support.

Last, but not least, I thank the Scientific and Technological Research Council of Turkey for their financial support in the project “Development of Personal Computer-based Motion Controller Systems” (no. 108E048), which has made it possible for me to conduct this thesis work.

TABLE OF CONTENTS

ABSTRACT iv
ÖZ v
ACKNOWLEDGEMENTS vii
TABLE OF CONTENTS viii
LIST OF TABLES xii
LIST OF FIGURES xiii
LIST OF SYMBOLS xvi
CHAPTERS
1 INTRODUCTION 1
1.1 Introduction 1
1.2 Simulation 2
1.3 Hardware-in-the-Loop Simulation 3
1.4 Objective of the Thesis 6
1.5 Organization 6
2 LITERATURE SURVEY 8
2.1 Introduction 8
2.2 Studies and Applications 8
2.3 Commercial Products 11
2.4 Dynamic Systems Modeling 13
2.5 Closure 16
3 MODELING AND SIMULATION TECHNIQUES 17
3.1 Introduction 17
3.2 Modeling Dynamic Systems 17
3.2.1 Transmission Elements 18

3.2.1.1 Gears 18
3.2.1.2 Rack and Pinion 21
3.2.1.3 Ball Screw 23
3.2.1.4 Belt 25
3.2.2 Power Generation Elements 27
3.3 Friction Modeling 30
3.4 Solver Techniques 36
3.4.1 Euler's Method 38
3.4.2 Heun's Method 39
3.4.3 Runge-Kutta Method 40
3.4.4 Adams-Bashforth Method 40
3.5 Creating Discrete-Time Models of Dynamic Systems 41
3.6 Closure 43
4 HARDWARE-IN-THE-LOOP SIMULATION FOR EDUCATIONAL APPLICATONS 44
4.1 Introduction 44
4.2 Proposed System 45
4.3 Application 48
4.3.1 Plant Model 50
4.3.2 Disturbance Model 50
4.3.3 Software Details 55
4.3.4 Hardware Platform 59
4.3.5 Interfacing 60
4.4 Results 60
4.5 Closure 64
5 REAL-TIME HARDWARE-IN-THE-LOOP SIMULATION UTILIZING FPGA 67
5.1 Introduction 67
5.2 Application 68
5.2.1 Plant Model 68

5.2.2 Hardware Platform 71
5.2.3 Interface Emulators 72
5.2.3.1 Incremental Encoder emulator 72
5.2.3.2 PWM receiver 75
5.2.4 Embedded Microprocessor Implementation 76
5.2.5 Parallel FPU Implementation 77
5.3 Results 78
5.4 Closure 84
6 NON-REAL-TIME HARDWARE-IN-THE-LOOP SOLUTION UTILIZING A HYBRID ARCHITECTURE 86
6.1 Introduction 86
6.2 Modeling 90
6.2.1 Plant Model 90
6.2.2 Disturbance Model 97
6.3 Implementation 100
6.3.1 PC Software 100
6.3.2 NGW100 Software 103
6.3.3 FPGA Design 106
6.4 Results 107
6.5 Closure 114
7 CONCLUSION AND FUTURE WORK 117
7.1 Conclusion 117
7.2 Future work 119
REFERENCES 121
APPENDICES
A HARDWARE SPECIFICATIONS 133
A.1 PIC Development Board and Microchip PIC Microcontrollers 133
A.2 NGW100 Network Gateway Kit and AT32AP7000 Microprocessor 134
A.3 Terasic Altera DE1 Cyclone II FPGA Starter Kit 136
A.4 Terasic Altera Nios II Embedded Evaluation Kit 137

B SOURCE CODE LISTINGS 139
B.1 Introduction 139
B.2 Source code – MainForm.cs 140
B.3 Source code – Antenna.cs 142
B.4 Source code – Environment.cs 148
B.5 Source code – Satellite.cs 149
B.6 Source code – Simulation.cs 151
B.7 Source code – StateSolver.cs 154
B.8 Source code – Protoc16.h 155
B.9 Definition and pseudocode – PWM_Receive.v 155
B.10 Definition and pseudocode – Encoder.vhd 156
B.11 Source code – main.cpp 156
B.12 Source codes for NGW100 platform – CadmusDevice.h 159
B.13 Source code – CadmusDevice.cpp 159
B.14 Source code – CadmusManager.h 162
B.15 Source code – CadmusManager.cpp 163
B.16 Source code – DisturbanceSource.h 166
B.17 Source code – DisturbanceSource.cpp 166
B.18 Source code – EncoderCommander.h 167
B.19 Source code – EncoderCommander.cpp 168
B.20 Source code – Solver_RK4.h 169
B.21 Source code – Solver_RK4.cpp 169
B.22 Definition and pseudocode – NGW_Interface.vhd 170
B.23 Definition and pseudocode – InputSignaller.vhd 171
B.24 MATLAB script – hils_milling_distgen.m 171

LIST OF TABLES

TABLES

Table 4.1: Questionnaire results for educational applications 64

Table 4.2: Cadmus simulation times 65

Table 5.1: Offline simulation test results 79

Table 5.2: Sample application resource usage 79

Table 5.3: 4th Order Runge-Kutta method computation times 83

Table 6.1: Plant parameters for CNC machining center 97

Table 6.2: Hybrid solution FPGA resource usage... 114

Table A.1: Microchip PIC18F4520 microcontroller specifications...................... 134

Table A.2: Specifications of Cyclone II EP2C20F484C7 FPGA chip 137

Table A.3: Specifications of Cyclone III EP3C25F324 FPGA chip..................... 138

Table B.1: Source files developed for the applications... 139

LIST OF FIGURES

FIGURES

Figure 3.1: Gear pair 18

Figure 3.2: Backlash in a gear pair.. 20

Figure 3.3: Transmission force in dead-zone backlash model 20

Figure 3.4: Rack and pinion 22

Figure 3.5: Backlash in a rack-and-pinion 23

Figure 3.6: Ball screw assembly 23

Figure 3.7: Backlash between screw and nut in a ball screw. 24

Figure 3.8: Two-pulley belt drive 26

Figure 3.9: Two-pulley belt drive with spring model 26

Figure 3.10: Direct torque control 28

Figure 3.11: Typical torque capability curve for an electric motors 29

Figure 3.12: Friction as a function of displacement as defined in Dahl's model.... 32

Figure 3.13: Illustration of the bristle model for friction 34

Figure 4.1: Operational block diagram for Cadmus software 47

Figure 4.2: Representation of the antenna dish and the definition of angles 49

Figure 4.3: Functional block diagram of the Cadmus application project 49

Figure 4.4: Antenna dish inside wind flow 51

Figure 4.5: Wind angle of satellite dish 52

Figure 4.6: Drag force on antenna dish with respect to wind angle 54

Figure 4.7: Disturbance torque on antenna dish with respect to wind angle 54

Figure 4.8: Main tab of Cadmus Software 56

Figure 4.9: Settings tab of Cadmus software 57

Figure 4.10: View tab of Cadmus software 58
Figure 4.11: Microchip PIC development board 59
Figure 4.12: Lead/lag controller design for satellite tracking antenna 60
Figure 4.13: Satellite and antenna azimuth angles for sample simulation using lead/lag controller 61
Figure 4.14: Motor and disturbance torques on azimuth axis for sample antenna simulation using lead/lag controller 62
Figure 4.15: Satellite and antenna elevation angles for sample simulation using lead/lag controller 62
Figure 4.16: Motor and disturbance torques on elevation axis for sample antenna simulation using lead/lag controller 63
Figure 4.17: Signal strength for sample antenna simulation using lead/lag controller 63
Figure 5.1: Hardware configuration for the FPGA solution 68
Figure 5.2: Spindle drive system 69
Figure 5.3: Block diagram of sample application system 70
Figure 5.4: Form of the disturbance torque applied on the spindle shaft 71
Figure 5.5: Terasic Altera DE1 Cyclone II FPGA Starter Kit 72
Figure 5.6: Incremental encoder output signals for clockwise rotation 73
Figure 5.7: Incremental encoder output signals for counter-clockwise rotation 73
Figure 5.8: Incremental encoder emulator block diagram 74
Figure 5.9: Illustration of timing in encoder emulator operation 74
Figure 5.10: Block diagram of PWM receiver 75
Figure 5.11: Modified PWM with synchronization signal 76
Figure 5.12: Block diagram for parallel-FPU CCDE solver 78
Figure 5.13: Computation time by CCDE degree, one equation per loop 81
Figure 5.14: Computation time by CCDE degree, two equations per loop 81
Figure 5.15: Parallel-FPU implementation computation time by CCDE degree 82
Figure 5.16: Parallel-FPU implementation resource usage by CCDE degree 82
Figure 5.17: Hysteresis and fuzzy controller performances for spindle drive 83

Figure 5.18: PID and sliding mode controller performances for spindle drive....... 84
Figure 6.1: Atmel NGW100 Network Gateway Kit 87
Figure 6.2: Block diagram of the microprocessor-based simulator 89
Figure 6.3: Operation timeline for microprocessor-based solution......................... 90
Figure 6.4: X-axis feed drive for CNC machining center....................................... 92
Figure 6.5: Z-axis feed drive for CNC machining center 92
Figure 6.6: First MCV-1100 3-Axis CNC Machining Center 93
Figure 6.7: Horizontal axes of the CNC machining center..................................... 94
Figure 6.8: Vertical axis of the CNC machining center.. 95
Figure 6.9: Torque capability curve for CNC machining center axis motors......... 96
Figure 6.10: Example of light cutting force disturbance on single axis.................. 99
Figure 6.11: Screenshot from the System Maker tool.. 101
Figure 6.12: Screenshot from the System Monitor tool....................................... 103
Figure 6.13: Mean times for sub-processes within the NGW100......................... 108
Figure 6.14: Cutting tool trajectory for bottle injection mold............................... 108
Figure 6.15: Section of axis references for bottle injection mold......................... 109
Figure 6.16: Section of *x*-axis motor position error for bottle injection mold...... 109
Figure 6.17: Section of *y*-axis motor position error for bottle injection mold...... 110
Figure 6.18: Section of *z*-axis motor position error for bottle injection mold 110
Figure 6.19: Section of *x*-axis cart position error due to backlash for bottle injection mold.. 111
Figure 6.20: Section of *x*-axis cart position error due to backlash for bottle injection mold.. 111
Figure 6.21: Section of *x*-axis cart position error due to backlash for bottle injection mold.. 112
Figure 6.22: Hybrid solution FPGA utilization floor plan.................................... 113

LIST OF SYMBOLS

Latin Symbols

A	antenna dish frontal area
A_d	antenna dish drag area
A_s	antenna dish side area
C_d	drag coefficient
b	viscous friction coefficient
D	backlash amount
F_d	drag/disturbance force
F_f	dry friction force
F_s	stiction force
F_t	backlash transmission force
F_{tot}	total force
h	simulation step size
h_s	ball screw lead
J_{eq}	equivalent inertia
J_m	motor inertia
k	spring constant
m_T	slope of torque limit in constant torque region
m_w	mass of workpiece
N	number of gear teeth, normal force
N_c	number of cutting edges on machine tool
P_r	rated power

T_d	disturbance torque
T_f	static friction torque
T_m	applied motor torque
T_m^*	reference motor torque
T_{max}	motor torque limit
T_r	rated torque

Greek Symbols

η_s	ball screw efficiency
μ	dry friction coefficient
θ	antenna elevation
θ_R	elevation reference
ψ	antenna azimuth
ψ_R	azimuth reference
ω	angular speed
ω_r	rated angular speed
ω_p	maximum angular speed

Acronyms and Abbreviations

ADC	Analog/Digital Converter
CCP	Capture/Compare/Pulse Width Modulation
CPU	Central Processing Unit
CTBGA	ChipArray Thin Core Ball Grid Array
CMOS	Complementary Metal-Oxide-Semiconductor
CNC	Computer Numerical Control
CCDE	Constant Coefficient Difference Equation
DMIPS	Dhrystone Million Instructions Per Second
DSP	Digital Signal Processor

DAC	Digital/Analog Converter
DMA	Direct Memory Access
DTC	Direct Torque Control
DDR	Double Data Rate
EEPROM	Electrically Erasable & Programmable Read-Only Memory
EMF	Electromotive Force
ECCP	Enhanced Capture/Compare/Pulse Width Modulation
EUSART	Enhanced Universal Synchronous/Asynchronous Receiver/Transmitter
FPGA	Field Programmable Gate Array
FTP	File Transfer Protocol
FIFO	First In First Out
FPU	Floating Point (Arithmetic) Unit
GPIO	General Purpose Inpt/Output
GUI	Graphical User Interface
GPU	Graphics Processing Unit
HILS	Hardware In The Loop Simulation
HTTP	Hyper Text Transfer Protocol
ISP	In System Programming
IrDA	Infrared Data Association
IO	Input/Output
IDE	Integrated Development Environment
I2S	Integrated Interchip Sound
I2C	Inter-Integrated Circuit
JTAG	Joint Test Action Group
LCD	Liquid Crystal Display
MAC	Media Access Control
MII	Media Independent Interface
MMU	Memory Management Unit
MIPS	Million Instructions Per Second

MMC	Multimedia Card
NASA	National Aeronautics and Space Administration
NGW	Network Gateway
NC	Numerical Control
OS	Operating System
PSP	Parallel Slave Port
PC	Personal Computer
PLL	Phase Lock Loop
PLC	Programmable Logic Circuit
PPR	Pulse Per Revolution
PWM	Pulse Width Modulation
QCIF	Quarter Common Intermediate Format
QVGA	Quarter Video Graphics Array
RAM	Random Access Memory
RISC	Reduced Instruction Set Computing
RMII	Reduced Media Independent Interface
SD (Card)	Secure Digital Card
SPI	Serial Peripheral Interface
SIMD	Single Instruction Multiple Data
SRAM	Static Random Access Memory
SMA	SubMiniature version A
SVGA	Super Video Graphics Array
SDRAM	Synchronous Dynamic Random Access Memory
SSP	Synchronous Serial Port
SSRAM	Synchronous Static Random Access Memory
SOPC	System On Programmable Chip
USB	Universal Serial Bus
USART	Universal Synchronous/Asynchronous Receiver/Transmitter
UAV	Unmanned Aerial Vehicle
VGA	Video Graphics Array

CHAPTER 1

INTRODUCTION

1.1 Introduction

Control systems are an inherent part of modern life. From the earliest ages of human history, timing and positioning mechanisms have been in use. Machines designed in the industrial era commonly employed centrifugal speed regulators, cams and other mechanisms to provide desired motion and synchronization. Over the decades, improving mechanical and electrical theory and practices, as well as emergence of the electronics discipline the fields of control theory and control engineering developed rapidly. Systems and machinery in the industry became more and more inseparable from control systems. Through improving technology, miniaturization and changing consumer needs, even common household items came to use controllers for their operation. Today, motor controllers, PLCs and various microcontrollers provide the necessary control and coordination for almost any production tool. From the provision of basic services to consumer electronics; microcontrollers, DSPs and logic circuitry of varying capacities take their places in every area of human life without us even noticing.

The importance of control systems in maintaining and advancing the modern living has led to massive amount of studies being conducted in the field. First formal analysis of centrifugal governors by Maxwell [1] in the late 19th century, followed by Routh's [2] generalization of the work to linear systems, great interest on the topic was raised in the academic field. People such as Lyapunov, Nyquist, Bellman and Ragazzini developed a number of techniques for analysis and design of dynamic systems and controllers. As a result, there exists an arsenal of tools, such as the Laplace and z- domains, root loci, Bode plots, state space design and so on are available to control engineers. Fields of non-linear, optimal and adaptive control are well established and advancing, while intelligent control elements like artificial neural networks and fuzzy logic controllers have gained widespread use both on their own and combined with other well-known techniques during last decades. Lately, the design of control systems have changed partially or completely into computer aided design processes, and progressing through computer automated design [3].

1.2 Simulation

With all the tools available to control engineers today, design of a system capable of accomplishing a given control task is not very hard. Clearly documented procedures and powerful computer tools relieve the designers from the burden of repeated calculations and allow them to focus on more important qualities of the design.

More often than not, the techniques used in the design process involve assumptions and simplifications in their theoretical bases. Similarly, the mathematical models of systems in question, which are needed for proper design, contain certain simplifications due to some parameters not being known, too complicated to model precisely or simply because the literature lacks the proper tools. Therefore the developed controllers, while good at fulfilling the requirements, are not perfect.

The increased expectations of today’s world, however, demand precision and perfection. To overcome the deviations due to imperfect design tools, control engineers are also equipped with a variety of inspection and analysis tools that help them investigate the controller performance. Analytical tools, combined with the iteration power of modern computers, allow the engineers to rapidly examine the system and controller behavior and fine tune their designs to meet specifications.

A very powerful and valid tool in exploring plant and controller performance is simulation. In simulations, numerical integration methods are used to determine the plant behavior based on system states and inputs. These methods not only free the analysis process from most simplifications and approximations introduced by other analytical tools, but they also allow the investigation of plants that include non-linear components (which are very difficult to cope with using traditional approaches) or those that act on discontinuous equations. Through the use of proper engines, even systems that require tools other than differential equations (e.g. state machines) to model can be simulated for investigation. With the computational power of modern computers, performing the calculations for simulations are easily done, making them a widely available tool.

The power of simulations is not limited to their ability to work with novel plant models. They also grant the developer the ability to monitor the states of any simulated component inside the plant without any additional computation. What's more, it is possible to directly manipulate these states (mimicking disturbances, malfunctions or similar internal or external effects) as well as the most basic system properties (such as masses, friction coefficients etc.) during the course of simulation. This way, introduction of deterministic and (pseudo) random disturbances, component degradation, changes in external circumstances and similar phenomena become possible. These opportunities cannot be provided by most traditional analysis techniques.

1.3 Hardware-in-the-Loop Simulation

In many cases, it is not possible to perform extensive investigation and testing of the mathematical qualities and topology of the controller. The actual controller device needs to be manufactured and issues related with its physical implementation should be taken care of. The necessary circuit design and assembly need to be done. The hardware and software programs that will realize the designed control algorithm should be developed and implemented. Any problems that may arise during sensor and actuator communication should be fixed. During these efforts, any possible constraints due to available hardware resources need to be met. The best and most valid way of accomplishing these goals is to couple the controller device with the plant to be controlled, providing all the conditions of actual operation.

In some cases, however, an implementation and testing process involving the actual system to be controlled may be undesirable. First of all, the cost of performing test runs on the plant might be too much. The cost of the plant itself, combined with the risk of damaging the equipment (or more importantly, the working personnel) in case of an error or malfunction may render the tests unfeasible. Secondly, collecting detailed information on a running plant is not an easy task. While some information can be obtained from the sensors already installed for normal operation, should more detailed information regarding other components be needed, fitting the system with necessary sensors requires careful planning and execution. Selection of suitable sensors, careful placement and mounting, proper laying and shielding of cables, use of capable data collection devices and so on need to be taken into consideration. Finally, the plant may not be even available for the engineer to work on. In tight development schedules, the development of the control systems may be required to be completed before a prototype of the plant is unavailable.

To overcome such difficulties, the concept of hardware-in-the-loop simulation ("HILS", also referred to as "HLS", "HIL simulation" or "HWIL simulation") is proposed. Basically, HILS is a technique where a controller device is connected to a simulation of the system to be controlled, via the emulation of sensor and actuator interfaces.

A powerful computing platform (which may actually consist of one or more devices) runs a high fidelity simulation of the plant in question. Facilities for controlling the simulation, monitoring system states, introduction of external effects on the simulation, injection of disturbances and recording of all the data generated during the process are provided via this platform.

An interface emulator stands between the simulator platform and the controller. This emulator provides the physical connection of the controller to the plant. Electrical signals that would be generated by the sensors are mimicked based on the simulation states, while command signals intended for driving the actuators are received from the controller and passed to the simulation as inputs. With the aid of this interface, the simulation is indistinguishable from the actual plant as far as the controller's point of view is concerned. The controller operates normally as if it would during normal operation without any significant modification.

The simulation might be performed in real- or non-real-time. In real-time simulations, the calculations are performed within a certain time frame. The simulation loop needs to be completed and necessary sensor data should be made available to the controller before it begins the execution of control calculations for the next sampling period. (It is necessary to note that the term "real-time" does not necessarily mean "high computational performance"; it rather indicates deterministic computation timings.) On the other hand, in a non-real-time simulation, such a timing constraint does not exist. Although the value of the sampling time inside the simulation and control calculations are kept the same, the actual execution time might be longer or shorter. The time may be scaled or the process may be carried based on a timing signal (which may not even have a set frequency) provided by the controller. In certain non-real-time simulations, where the timings of the signals transmitted between the controller and the simulation are

meaningful, the interface emulation may also need to be adjusted to make up for the change in actual execution time.
Hardware-in-the-loop grants the control engineer the chance to work with the almost-final design of the controller. With all the possibilities provided by running a simulation, almost any situation that can be encountered during normal operation can be generated and necessary changes on the design can be made.

1.4 Objective of the Thesis

The goal of this thesis is to develop a hardware-in-the-loop simulator. This system should be able to interface with a controller device using communication methods commonly found in such devices. It should also be able to work with any plant that the user specifies by defining its mathematical model. Finally, the data generated during simulation should be recorded and presented to the user in a convenient form for analysis and optimization purposes. One desired property of the solution is that it should not require any additional hardware or commercial software packages other than those provided in it. In other words, the proposed solution should be integrated and self-sufficient for its purposes.

To accomplish the goals set here, mathematical tools, hardware and software that are suited for performing HILS will be investigated. Different approaches to HILS will be explored, their qualities will be inspected and the results will be compared.

1.5 Organization

Following the introduction chapter, Chapter 2 gives information on the current state of the art in hardware-in-the-loop simulation applications. Various applications and products for HILS are reviewed. A brief survey regarding modeling techniques for dynamic systems is presented.

Chapter 3 discusses the mathematical models for components that are commonly encountered in dynamical systems are presented. Numerical integration techniques and other solver routines for carrying out the simulation operations are given.

Chapter 4 proposes a non-real-time HILS solution for use in education of control engineering. Application of this solution in an actual course is explained with results.

Chapter 5 gives details on a general-purpose, real-time HILS system. Components for enabling simulation-controller interfacing are also presented.

Chapter 6 develops an HILS system consisting of multiple hardware platforms, complete with design and execution tools.

Finally, Chapter 7 summarizes the results of the study and hints at possible future developments.

CHAPTER 2

LITERATURE SURVEY

2.1 Introduction

HILS is a powerful tool that has found applications in a wide variety of areas, from marine research to space missions and even education. In order to set the background and current state of hardware-in-the-loop simulation and relevant systems, this chapter explores and presents information on the various studies and applications on the subject. Commercially available hardware and software products enabling or aiding HILS applications are also given here. Finally, a brief survey regarding the modeling of components found in dynamical systems is presented.

2.2 Studies and Applications

The use of HILS has a history dating back over 40 years. Driven by military purposes where demands for precision are high due to involved risks, simulation of tactical missiles for guidance systems testing constitutes examples of some of the earliest works. The Sidewinder missile program is one example of such works employing HILS during the late 1960's. Powerful hardware have been used in real-time 6 degree-of-freedom simulation of active missiles and targets [4]. Radio frequency and millimeter-wave radar signal injectors, infrared image and electro-optical signal generators are used to simulate the sensor inputs to the missiles [5-6]. Instead of modeling sensors along with relevant noise and error forms, such an approach provides highly realistic simulations. Also, the opportunity to test the entire control system hardware rather than only the processor is thus provided [7].

Development and production of the instrumentation, equipment and various systems used in aerospace are quite demanding. Due to the high amount of funds spent for research in the field and the risk of accidents involving irreversible damages, loss of equipment or even human lives; extensive testing of all the systems used is required. HILS is therefore invaluable for this field. The developers of the highly maneuverable aircraft technology, or HiMAT, remote-piloted vehicles benefitted greatly from this tool [8]. The Cassini Spacecraft and its mission bound for Saturn is a good demonstration of space applications employing HILS [9]. A multitude of systems, such as the attitude and articulation control, command and data subsystem and many other components were tested using high-fidelity simulations. In addition to verification of sequences, procedures and software; the simulations have also served in training the crew on duty in the mission. Another application of HILS on the special purpose dexterous manipulator used in the International Space Station includes simulation of numerous systems involved in the system [10]. A novelty of this study is the employment of a scaled rigid robot that is used instead of the mathematical model of contact dynamic models. Investigation of the kinematics and dynamics of the manipulator on a space satellite system with vision sensor was done by Chinese researchers [11]. The Formation Flying Test Bed developed by NASA uses dynamic HILS of the guidance, navigation and control analysis for clusters of satellites maintaining a formation [12]. In another research, dynamical simulation of picosatellite sensor nodes in low earth orbit was performed for a distributed orbital computer network intended for use in space missions [13].

Verification of structures and controllers of unmanned marine vessels, both surface and underwater, may be done using HILS. Navigation algorithms and other software, maneuverability, energy consumption and dynamical qualities of the vehicle such as stability and maneuverability are investigated using simulations [14-15]. In a work by researchers, hydrodynamics, models of thrusters, propellers and various control surfaces, waves and currents in the water and other similar effects were included in the simulations [16]. Aside from individual vessels, a simulation architecture for cooperative operation of multiple unmanned vehicles was also proposed [17]. This architecture involves environmental emulation including acoustic propagation model, virtual sensors and communication devices.

Today, use of HILS in the automotive industry has become commonplace. Tight budgets and deadlines, as well as demanding safety measures require the engineers to extensively test the performance and reliability of numerous mechanical elements and embedded controllers. For this purpose, simulation platforms aiming for testing different systems were developed by many different facilities [18-24]. The validation of many safety and driving aid systems are also done using HILS [25-26]. To meet the high production rates and extensiveness of the validation procedures involved, work has been done on automation of the testing process [27]. Latest work in the field not only involves simulation of the mechanical and electrical systems, but also the injection of faults into these in order to explore their performances under imperfect conditions [28].

HILS has received attention in testing and validation of electrical machinery and power electronics systems as well. Researchers have proposed mathematical tools, procedures and implementation methods for realizing such simulations [29-30]. Applications regarding the simulation of electronic components such as converters and filters were developed [31-33]. Simulations of electromechanical drives using different methods such as finite element models were also presented [34-35].

The plants that can be simulated by HILS techniques are not limited to dynamical systems or those that are defined using differential equations. Plants governed by logical rule sets or discrete equations, as well as networks of plants can also be simulated. Communications experts have employed HILS in analysis of mobile devices belonging to various-scale wireless communication networks [36-37]. Models of undersea networks, communication backbones and ocean observatories were developed and investigated by ocean researchers [38]. Numerous works on analysis and optimization of traffic signal controllers were carried out [39-42].

The conveniences provided by HILS have made it a feasible option in education, too. Teachers aiming to provide low-cost and high-efficiency laboratory work for the students have resorted to such applications [43-45].

2.3 Commercial Products

Today, HIL simulations aren’t only encountered in research studies and special projects. While the developing technology lowers the cost of powerful hardware, increasing customer expectancy for everyday product performance and reliability has introduced medium and even small scale developers and manufacturers with the necessity for simulations. To close the gap between these parties and the already-advanced HILS technology, several other hardware and software developers have stepped in.

Speedgoat [46] is a manufacturer for highly-flexible, scalable hardware platforms for use in HILS applications. Processor platforms of varying cost and performance levels are provided. Numerous expansion modules for analog and digital I/O, serial and parallel communication and working with pulse-based signals (encoder/decoders, PWM etc.) are available. Shared memory modules can be used for data transfer between multiple products or third-party devices. FPGA-based configurable modules for further customization purposes also exist.

dSpace [47] provides hardware and software products for automotive, aerospace and industrial control applications. Hardware platforms for both implementing controllers and performing their HILS testing are available. Extension modules for extensive controller communication and calibration, as well as data diversion and bypassing for advanced testing applications, can be used. To aid the development process, a variety of specialized software packages are also provided. Design and implementation of controllers, experimentation and testing, generation of software embedded in the products, measurement, calibration and simulation tasks are facilitated by these packages.

Opal-RT [48] is mainly a software developer focusing on real-time simulator packages specifically tailored for a multitude of fields, as well as toolboxes for use with other existing design and analysis software. Controller prototypes, motor drives, converters and various scale electricity grids are covered by the electrical & power systems simulators. Aerospace & defense product family provides tools for simulating various flight systems, turbine engines and UAVs. Complete automobile simulation including electric and hybrid cars, engine and transmission assemblies, and complete including hybrid and electric automobiles including hybrid and vehicle dynamics are made possible by the automotive products. Along with all the software tools, an array of I/O and signal conditioning modules are also provided for controller interfacing.

Applied Dynamics International [49] is the developer of the ADvantage Framework, a comprehensive set of software tools for real and non-real-time HILS, as well as distributed simulation. The framework provides both development and run-time environments for these applications. The Beacon product family provides automated generation of safe and reliable controller code generation, as well as providing aid in development of test cases for the generated code. A line of hardware products, Emul8, are also available for supplementing the simulation of automobiles or military vehicles, for the purpose of designing automotive electronic control modules.

Aside from tools specifically designed for HILS, other companies have developed additional modules for their products to enable such simulation applications. National Instruments [50] provides the NI HIL Simulator Reference System line of hardware products and associated plug-ins for the LabVIEW software package. Similarly, The MathWorks [51] proposes the xPC Target Turnkey hardware family along with the necessary libraries for Matlab/Simulink software packages. Both companies supplement their products with a variety of I/O modules.

2.4 Dynamic Systems Modeling

Proper modeling of the plant is essential in all simulation applications. Quite often, dynamics of the systems are not limited to simple forces and masses, but are instead composed of many different subsystems governed by a large variety of phenomena. To obtain an acceptable fidelity, these should be investigated and modeled using appropriate techniques and approximations.

Friction is an effect that occurs in all mechanical systems, appearing at the physical interface between two contacting surfaces. As the need for reducing and compensating friction increased by the ever-demanding market, friction has received great attention by researchers. Early documented work on friction dates back to mid-18th century by Coulomb on dry friction and late 19th century by Reynolds [52] on viscous friction. Later in early 20th century, work by Stribeck [53] gives the friction force as a function of velocity itself in constant-velocity motion, with a sudden drop at lower speeds (also known as the Stribeck effect). The *stiction* phenomenon, which is the existence of friction force higher than the Coulomb friction when the body is at rest, is explained by Morin [54]. In modeling the sticking effect and break-away in static friction, Rabinowicz [55] addresses the transition process as a function of displacement. Johannes et al. [56] and Richardson et al. [57] indicate in independent works that the magnitude of the force required to overcome sticking condition depends on the rate of application of the force. Courtney-Pratt et al. [58] suggest a spring-like behavior in relative displacement of two bodies before motion occurs. In more recent work, Karnopp [59] proposes a method for addressing the issue of detecting zero velocity in computer simulations using a dead band zone. Armstrong et al. [60] later introduce temporal dependencies to the classical friction model to reflect certain observed dynamic effects.

With the availability of more powerful hardware and the demand for high precision in servo motors, dynamic models for friction have received great interest. Driven by experiments on servo systems with ball bearings, Dahl [61] models the friction force as a function of displacement (using a differential equation that is based on the stress-strain curve in solid mechanics) and also proposes a time domain model that is a generalization of Coulomb friction. The absence of Stribeck effect and stiction in Dahl's equations later leads to the extension of this model by Bliman [62]. In an attempt to capture the effect of irregular micro-scale contact between two surfaces, Haessig et al. [63] introduce a model based on the bending, snapping and random re-bonding of flexible bristles between contacting surfaces. Although inefficient for simulations, this model provides good results that capture the random nature of friction. In the same work, a modification is made on the model by adding another state in determining the strain of the bristles, in an attempt to come up with a more computationally feasible "reset integrator" model. Bliman et al. [64-66] propose a linear model in state-space, partially exhibiting the stiction, break-away and Stribeck effects, which reduces to Dahl's model when expressed in its first order of complexity. An extensive model that is based on the bristle model and covers the Stribeck effect, break-away and stiction is proposed by Canudas de Wit et al. [67] with the name "LuGre friction model".

Belts are widely used in a variety of applications requiring the transmission of power over a distance. Early modeling of belt mechanics is presented by Reynolds [68] and further developed by Swift [69] under the "creep theory". Their works depend on the idea that frictional forces exerted on the belt by the pulleys cause the elastic belt to extend and contract. This model is commonly applied using lumped parameters, by representing traversing belt parts as springs or spring-damper assemblies, in such works by Abrate [70] and Hace et al. [71]. Bechtel et al. [72] extend the creep theory to include the inertial effect on the part of the belt wrapped around the pulleys. Work by Rubin [73] applies the model to multi-pulley drives that are commonly used in many applications. Firbank [74] claims that creep theory is inadequate in modeling modern belts that contain high-stiffness steel fibers, instead proposing the “shear theory”. This model explains the belt transmission based on the shear strains between the pulley surface and the cords within the belt. Work by Childs et al. [75] confirm the shear theory by the power loss in belt drives. Alciatore et al. [76] show that this model is necessary to handle multi-pulley cases with inextensible belts. Gerbert [77-78] extends the shear theory to flat, V and multi-ribbed V belts.

Backlash, which is the undesired clearance between teeth of mating gear pairs, is a common effect deteriorating the control performance of a system. Therefore it is necessary to obtain a suitable backlash model for effective compensation. Slotine [79] proposes a dead zone model, which has become the most common approach to backlash. Sarkar et al. [80] model backlash as an impact event that occurs when mating gears come into contact. An exact backlash model is proposed by Nordin et al. [81].

Coming in many different sizes, shapes and characteristics; today electric motors are the primary source of actuation in almost any dynamical system. Due to their high power output, AC motors are commonly used in industrial applications. However, since these motors are governed by complex electromechanical processes, it is desirable to add a layer of control to achieve a control scheme that aims to obtain a desired output from these machines. For this purpose, Yamamura et al. [82] propose the field acceleration method. Claiming that it is often difficult to force the desired currents or voltages into stator windings, Depenbrock [83] develops the concept of “direct self-control”, later to be known as “direct torque control”. Using flux and torque estimators, French et al. [84] develop and implement a DSP-based direct torque controller. Mir et al. [85] utilize PI and fuzzy estimators to implement a space vector modulation scheme for realizing DTC. Behera et al. [86] propose and validate a dither injection method to reduce torque ripple in DTC. In an attempt to describe the characteristics of induction motors, Soong [87] investigates their field weakening performance and gives the torque-speed curve commonly utilized in modeling motors. In later work, Moore [88] investigates the effects of extending the constant power region of the described curve.

2.5 Closure

In this chapter, background on the various applications of HILS was given. Some of the numerous fields of application were discussed, hinting at the specific application purposes. Information on a few select commercial products available for HILS was presented. A summary on modeling of dynamical systems was also given.

CHAPTER 3

MODELING AND SIMULATION TECHNIQUES

3.1 Introduction

The first and most important element in performing a simulation is the mathematical model of the dynamic system to be simulated. The fidelity of the simulation to the real world is very much based on the depth and accuracy of the models used. Careful examination and modeling of the components making up the plant is therefore required. However, most of the time, the workings of these components involve phenomena that are too complex to model or computationally costly. Therefore, approximate yet accurate models need to be developed. In this chapter, mathematical models for some components that are commonly found in dynamical systems are given. While the bases of the models are explained, complete derivations of the results are kept outside the scope of this thesis.

Another important part of a simulation is the solver routines. The suitability of the selected solver to the equations that govern the plant should be investigated. The step size and any other parameters (if applicable) should be determined correctly. Brief information on common solution methods are also presented in this chapter.

3.2 Modeling Dynamic Systems

Conversion and transmission of power make up for most of the processes that take place in dynamical systems. Modeling of these events therefore provides equations that can be reused in many different systems with minimal modification. The following sections derive the models for a handful of such commonly encountered components.

3.2.1 Transmission Elements

Due to geometrical constraints, weight or balancing concerns or other similar reasons; power cannot always be generated (or converted) at the location it is needed. Furthermore, the direction, speed, magnitude or range of the power source may need to be changed for the purposes of the application at hand. Therefore, transmission elements such as shafts, pulleys, belts, cables, gears and so on are commonly used to transmit and reshape the generated power. This section provides the mathematical models for some common transmission elements and/or phenomena associated with them.

3.2.1.1 Gears

Gears are rotating machine parts that make use of cut teeth to mesh with and drive each other in order to transmit power. Coming in a variety of designs, gears can couple shafts arranged in various setups: parallel, intersecting at an angle or skew. Combined with the ability to generate mechanical advantage, gears are preferred in a wide range of applications.

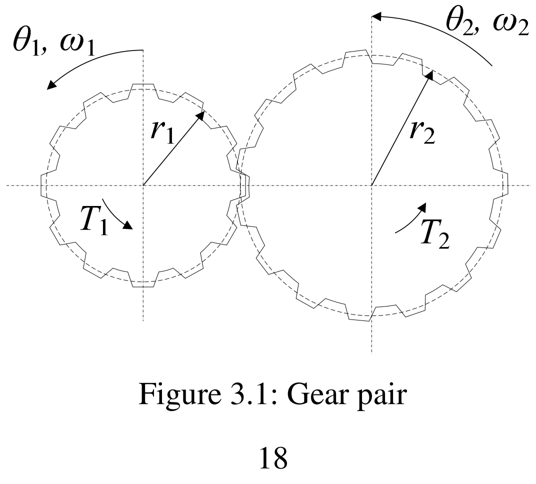


Figure 3.1: Gear pair

For ideally manufactured and assembled gear pairs, the torque T_2 applied by the driving shaft on the driven shaft is given by

$$T_2 = -T_1 \frac{N_2}{N_1} \tag{3.1}$$

where T_1 is the drive torque and N_1, N_2 are the number of teeth on the gears. The teeth number ratio is also called the gear ratio (N_g). The ratios of the angular positions θ_1, θ_2 and velocities ω_1, ω_2 of the shafts are

$$\frac{\theta_1}{\theta_2} = -\frac{N_2}{N_1} \tag{3.2}$$

$$\frac{\omega_1}{\omega_2} = -\frac{N_2}{N_1} \tag{3.3}$$

Despite the techniques used in manufacturing and assembling gears, non-ideal gear pairs are commonly encountered. Improper gear profiles, incorrect center distances and wear introduce excessive clearance between mating gears. This clearance leads to an effect called “backlash”. Defined as “the play between adjacent movable parts”, backlash can be described as the unwanted motion (or loss of motion) due to excessive clearance between contact surfaces. Backlash occurs when contact between the mating parts is lost and re-established because of changes in movement direction or speed. Aside from gears, backlash is also found in other contacting elements such as roller bearings or ball screws. Although methods have been devised to eliminate backlash (such as duplex worm gears or preloaded elements), these cannot be used in all applications and backlash is therefore commonly encountered.

A common method of modeling backlash is the dead-zone model. In this approach, the input and output gears are coupled together with a transmission force that is a function of the difference between the angular positions of the gears – or more precisely, the position difference along the pitch lines.

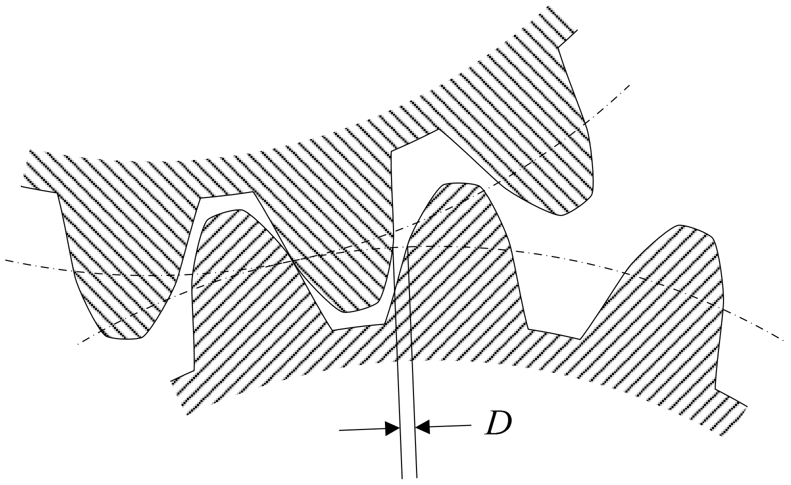


Figure 3.2: Backlash in a gear pair

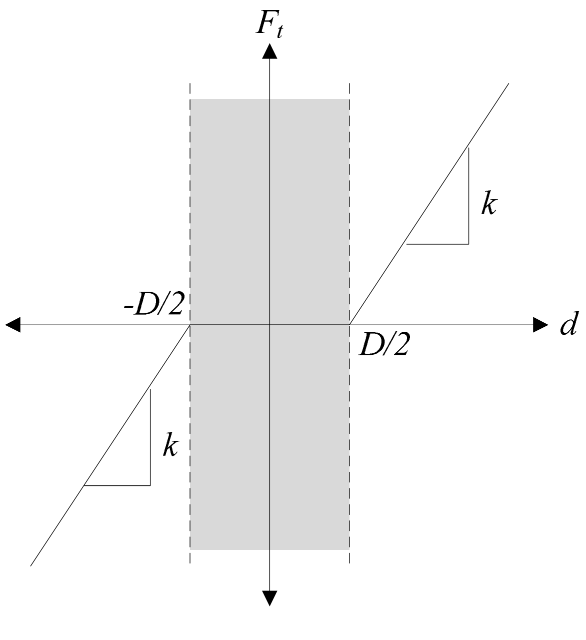


Figure 3.3: Transmission force in dead-zone backlash model

In Figure 3.3, k is a spring constant dependent on the material and geometry of the shaft and gear and D is the amount of backlash between gears. d is the position difference of the gears along the pitch line and can be calculated as

$$d = r_1\theta_1 - r_2\theta_2 \tag{3.4}$$

where r_1, r_2 are the pitch radii of the input and output gears, respectively. The transmission force F_t, in turn, is

$$F_t = \begin{cases} k\left(d - \frac{D}{2}\right) & , d > \frac{D}{2} \\ 0 & , |d| < \frac{D}{2} \\ k\left(d + \frac{D}{2}\right) & , d < -\frac{D}{2} \end{cases} \tag{3.5}$$

The torques applied on the gears are simply

$$T_1 = F_t r_1 \tag{3.6}$$

$$T_2 = F_t r_2 \tag{3.7}$$

It follows that while the positional difference of the gears is within the dead-zone, the shafts connected to them are uncoupled. However, when the difference moves outside the zone, the gears get into contact and the shafts act as if coupled via a torsion spring.

3.2.1.2 Rack and Pinion

Rack and pinion assemblies are used to convert rotational motion into translational motion using a gear and a rack with mating teeth, similar to a gear pair. Figure 3.4 shows the rack and pinion assembly.

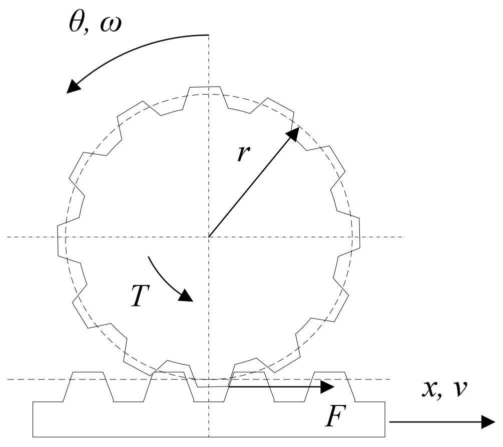


Figure 3.4: Rack and pinion

The force transmitted by the pinion on the rack is simply calculated from

$$F = Tr \tag{3.8}$$

And the relation between the positions and velocities are

$$x = \theta r \tag{3.9}$$

$$v = \omega r \tag{3.10}$$

The dead-zone backlash model can be easily extended into the rack and pinion assembly. Figure 3.5 shows the definition of backlash in a rack and mating pinion. The position difference of the rack and pinion is simply

$$d = r\theta - x \tag{3.11}$$

Equation (3.5) holds for the transmission force (which directly applies to the rack) and

$$T = F_t r \tag{3.12}$$

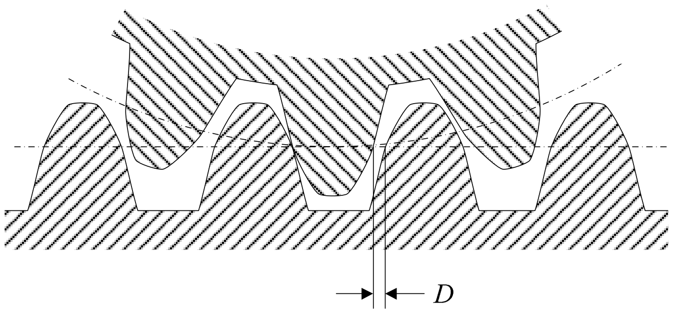


Figure 3.5: Backlash in a rack-and-pinion

3.2.1.3 Ball Screw

Ball screws are mechanical devices that are used to convert rotational motion to linear motion with minimal friction. They are similar to lead screws in the sense that they employ a driving threaded shaft and a driven nut. However, instead of coupling these elements directly, ball screws use a number of ball bearings to transfer force between them. This eliminates the sliding friction between the screw and nut and only introduces, rolling friction which is significantly lower, resulting in higher efficiencies (typically over 90%). Figure 3.1 shows a ball screw assembly where the nut, shaft, balls and the ball circulation mechanism are visible.

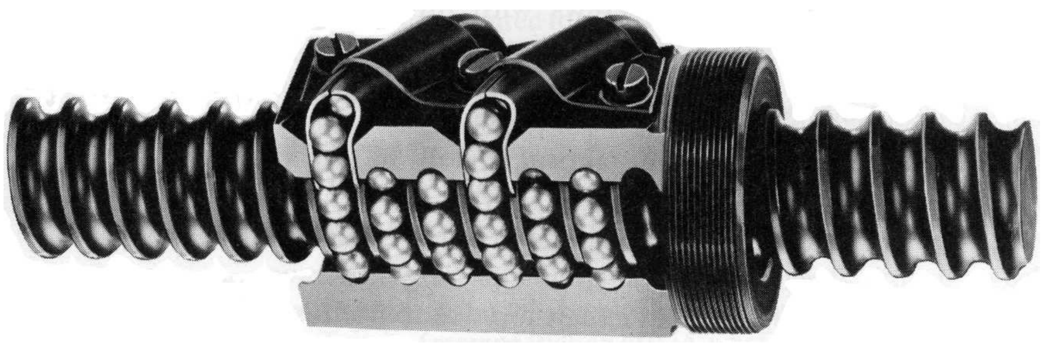


Figure 3.6: Ball screw assembly [89]

The force *F* applied on the ball screw nut by the shaft can be calculated as follows:

$$F = T\frac{2\pi\eta_s}{h_s} \tag{3.13}$$

where T is the torque applied on the shaft, h_s is the screw lead and η_s is the screw efficiency.

Similar to gear pairs, backlash also exists in ball screws. Figure 3.7 shows the view of a single ball inside the screw and nut assembly, with backlash indicated in two ends of the dead-zone. Here, the backlash is defined as $D = D_1 + D_2$. The position difference is

$$d = x - x_s \tag{3.14}$$

where *x* is the position of the nut along the shaft and x_s is given by

$$x_s = \theta\frac{h_s}{2\pi} \tag{3.15}$$

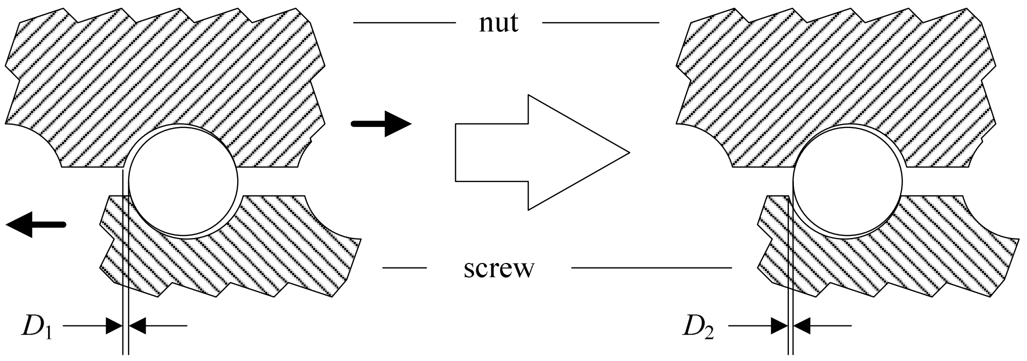


Figure 3.7: Backlash between screw and nut in a ball screw.

where θ is the angular position of the shaft. Equation (3.5) holds for the transmission force, applying directly on the nut and via

$$T = F_t \frac{h_s}{2\pi\eta_s} \tag{3.16}$$

on the screw.

3.2.1.4 Belt

From turning centers to automobiles, belts are commonly used in transmission of power over small to medium distances. Their high efficiency, tolerance to misaligned pulleys and low maintenance requirement make belts a desirable means of transmission. Although the angular velocity ratio of the driven pulley (or multiple pulleys) to the driver may deviate from the ratio of pulley radii due to slippage, toothed belts (or timing belts) solves this problem. Still, this ratio may oscillate because of the stretching of the belt.

Although idler pulleys (for adjusting belt tension) and driving of more than one pulley with a single belt are common, a belt connects one input pulley to one output pulley in the most basic case (Figure 3.8).
The mathematical model of such an assembly, in its simplest form, neglects the dynamics of the belt:

$$\frac{T_2}{T_1} = \frac{r_1}{r_2} \tag{3.17}$$

$$\frac{\omega_2}{\omega_1} = \frac{r_1}{r_2} \tag{3.18}$$

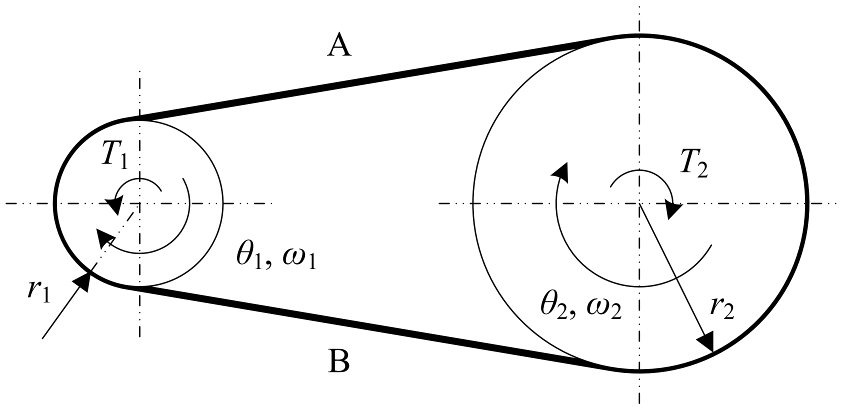


Figure 3.8: Two-pulley belt drive

This model of a belt only serves as a relation between the masses (shafts or other elements) connected to the pulleys. The position (as well as velocity, acceleration and jerk) and torque of the output can be calculated directly.

A model of the belt based on the creep theory includes the tension and compression (or bending) of the belt segments A and B on Figure 3.8 by treating these effects as springs (Figure 3.9).

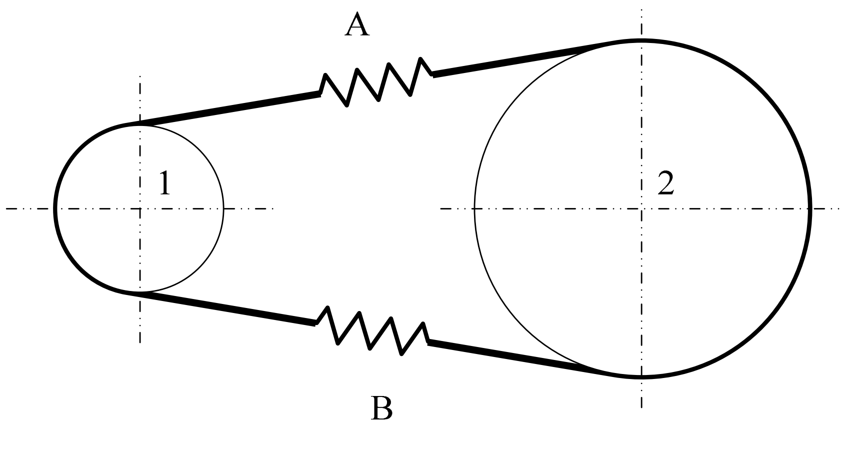


Figure 3.9: Two-pulley belt drive with spring model

In such a model, the input and output shafts are no longer simply coupled. Instead, the equations of motion need to be written separately for both. The equations of motion for the two shafts using this model, with pulley 1 being the driving side, are:

$$J_1\dot{\omega}_1 = T_1 - r_1 2 f_{spring} \tag{3.19}$$

$$J_2\dot{\omega}_2 = T_2 + r_2 2 f_{spring} \tag{3.20}$$

where

$$f_{spring} = k\left(r_1\theta_1 - r_2\theta_2\right) \tag{3.21}$$

From (3.19) through (3.21), it can be seen that the angular speeds of the pulleys will oscillate during solution unless damping torques act on them. Therefore, careful modeling of the other plant components is required.

3.2.2 Power Generation Elements

In almost all dynamical systems, some form of power (e.g. thermal, chemical or electrical) is converted into mechanical power and harnessed to accomplish the desired task. Of such converters, electric motors, which employ the magnetic field generated by electric current to convert electrical energy to mechanical energy, are the most widely used ones. Manufactured in a great variety of sizes and shapes, they can be found almost anywhere from ship propellers to wrist watches. The subject of this section, however, is AC servo (induction) motors that are commonly used in various industrial applications.

Driving an induction motor typically involves fine and high-bandwidth control of the currents flowing through the rotor windings. In most applications, however, it is convenient to manipulate the torque output of the motor rather than winding currents. Direct Torque Control (DTC) is a controller scheme widely used for this purpose. Utilizing flux and torque estimators that observe the winding currents, DTC provides a means for directly controlling the torque output of the motor using only a reference torque command (Figure 3.10).

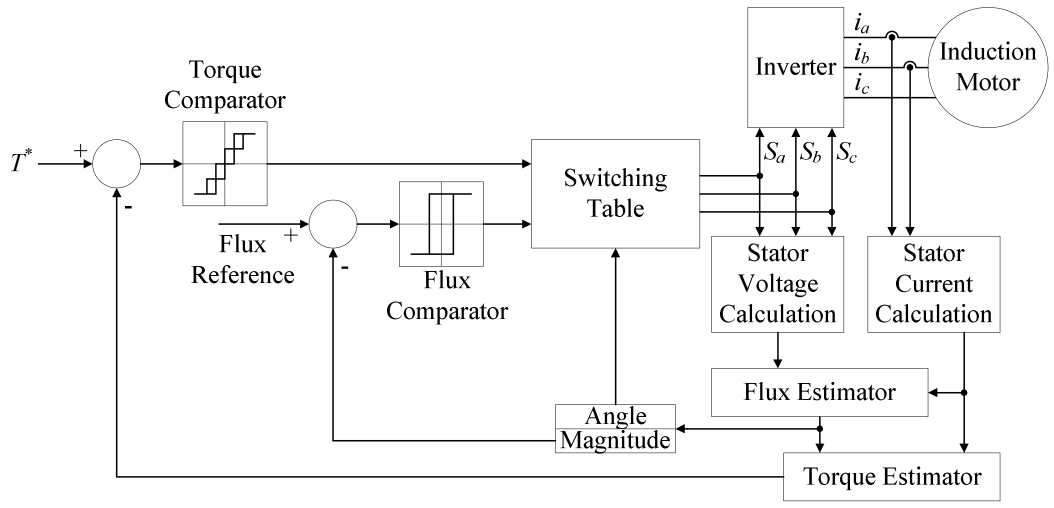


Figure 3.10: Direct torque control

While the use of DTC simplified an induction motor to a torque modulator, motors aren't capable of applying the desired torque output. Rather, they have a certain torque characteristic that limits the maximum applied torque based on the angular speed of the motor. This limit, the "torque capability curve", consists of three distinct regions as seen in Figure 3.11.

As can be inferred from the figure, a typical motor can deliver up to its rated torque (T_r) at all speeds up to its rated speed (ω_r). At this state, the motor output is at its maximum, which is called the rated power (P_r). Past the rated speed, the constant power region is entered. The maximum torque of the motor is inversely proportional to the speed in this zone, up to the maximum speed (ω_p). When the

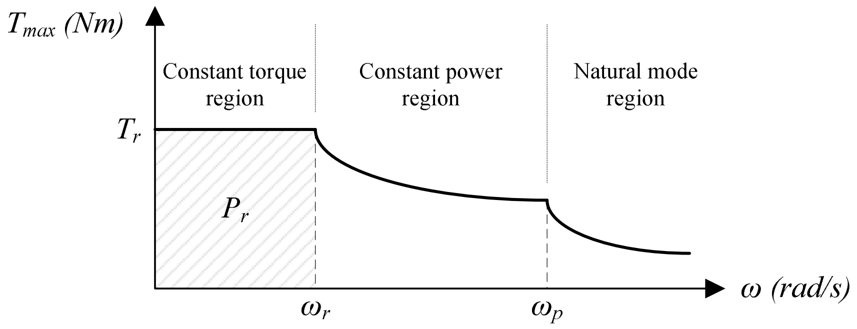


Figure 3.11: Typical torque capability curve for an electric motors

motor is pushed beyond this speed, the constant power region starts to degrade, entering the natural mode where the torque capability decreases proportional to the square of the speed. However, this zone is not used in most applications.

Assuming the utilization of DTC technique and neglecting the natural mode region, the motor can be modeled simply as a near-ideal torque modulator obeying the capability curve. The envelope of the torque capability curve (*T_{max}*) is defined by

$$T_{\max} = \begin{cases} T_r, & |\omega_m| \le \omega_r \\ \dfrac{T_r \omega_r}{|\omega|}, & \omega_r < |\omega| \le \omega_p \\ 0, & else \end{cases} \tag{3.22}$$

and finally the output of the motor (*T_m*) based on the reference torque (*T_m^**) is

$$T_m = \begin{cases} \mathrm{sgn}(T_m^*) T_{\max}, & T_m^* \ge T_{\max} \\ T_m^*, & else \end{cases} \tag{3.23}$$

3.3 Friction Modeling

One of the most common and important phenomena in dynamic systems is friction. It can be encountered in almost any component, from shafts to linear guideways, gear tooth interfaces and any other place where contact and relative motion exists. Basically, friction can be defined as the tangential reaction force between contacting surfaces. Physically, these forces can be the result of many different mechanisms, involving contact geometry and topology, bulk and surface materials of the bodies in contact, amount and rate of displacement and lubrication properties. Many of these mechanisms involve effects that act only under certain combination of conditions or for a limited time during operation. It is therefore not possible to construct a single, universal model taking all these effects into account. Therefore the trend has been to develop approximate models for certain application conditions.

Classical friction models consist of different components, each focused on a different aspect of the friction force. Their common basis is that friction always opposes motion and is independent of the contact area. The most basic classical friction model is the *Coulomb friction*, expressed as

$$F = F_f \operatorname{sgn}(v) \tag{3.24}$$

where F_f is proportional to the normal contact load;

$$F_f = \mu N \tag{3.25}$$

where μ is the dry friction coefficient. Coulomb friction is commonly used in controllers for friction compensation due to its simplicity. Another classical friction model covers the effects of the viscosity of lubricants, and is named *viscous friction*. This is normally described as dependent on the direction of motion by

$$F = b|v|^{\delta_v}\operatorname{sgn}(v) \tag{3.26}$$

where b is the viscous friction coefficient. δ_v is a positive constant given by the specific application and may be determined from experimental results. *Stiction* describes the friction force applied while the body is at rest, which is typically higher than the Coulomb friction. This force always counteracts the total external force applied on the body, keeping it at rest. Since no motion is related with stiction, it is a function of the force acting on the body:

$$F = \begin{cases} F_{tot} & , \ |F_{tot}| < F_s \\ F_s & , \ |F_{tot}| \geq F_s \end{cases} \tag{3.27}$$

where F_{tot} is the total external force on the body and F_s is the stiction force. It is important to note that (3.27) only holds as long as the body is at rest. In most modeling studies; the dry and viscous frictions and stiction effect are used together to yield the following single equation for friction:

$$F = \begin{cases} F_{tot} & , \ |v| = 0 \ \ and \ \ |F_{tot}| < F_s \\ F_s & , \ |v| = 0 \ \ and \ \ |F_{tot}| \geq F_s \\ \left(b|v|^{\delta_v} + F_f\right)\operatorname{sgn}(v) & , \ |v| > 0 \end{cases} \tag{3.28}$$

In simulation implementations of (3.28), it is important to take into consideration the fact that velocity will most probably fail to settle at exact zero, due to errors introduced by floating point arithmetic and/or controller effort. Therefore to detect stiction condition, as proposed by Karnopp, a small velocity range $[-\varepsilon, +\varepsilon]$ may need to be defined, inside of which the velocity is forcedly set to zero.

When more precise friction calculations are required, classical static friction models are insufficient and dynamic models should be used. An early dynamical friction model given by Dahl originates from the stress-strain curve in solid mechanics. When force is applied on the body, the friction force increases it reaches the rupture point. The stress-strain curve is modeled by the differential equation

$$\frac{dF}{dx} = \sigma\left[1 - \frac{F}{F_C}\operatorname{sgn}(v)\right]^{\alpha} \tag{3.29}$$

where F is the friction force, F_C is the Coulomb friction, σ is the stiffness coefficient and α is a parameter used to define the sharpness of the bend in the curve, most commonly taken equal to unity.

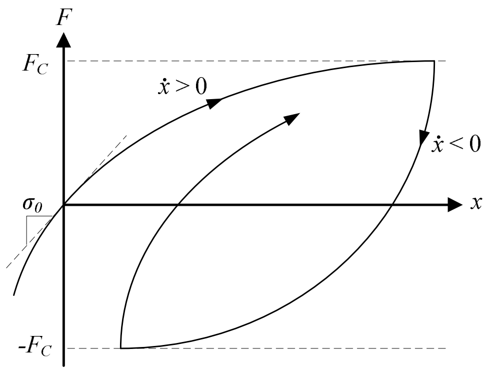


Figure 3.12: Friction as a function of displacement as defined in Dahl's model

It should be noted that equation (3.29) does not take the velocity into account, implying a friction force that is only dependent on the position. To obtain a time model, Dahl utilizes

$$\frac{dF}{dt}=\frac{dF}{dx}\frac{dx}{dt} \tag{3.30}$$

In turn, one obtains

$$\frac{dF}{dt}=\sigma\left[1-\frac{F}{F_C}\operatorname{sgn}(v)\right]^{\alpha} v \tag{3.31}$$

which is a generalization of the Coulomb friction.

Another dynamical friction model is the "bristle" model proposed by Haessig and Friedland. This model approximates the micro-scale sticking and separation between the sliding surfaces using a number of flexible bristles between the bodies (Figure 3.13). Every contact point between these bodies is represented by two bonded bristles. As the bodies move relative to the each other, these bristles bend and act as springs that cause the friction force. This force is calculated as

$$F=\sum_{i=1}^{N}\sigma_0\left(x_i-b_i\right) \tag{3.32}$$

where N is the number of bristles in the contact area, σ_0 is the bristle stiffness, x_i is the relative position of each bristle and b_i is the position where the bristle contacts are formed. As the bodies move, $\left|x_i-b_i\right|$ increases until it equals δ_s, at which point the bond is broken and a new bond is formed at a new location randomly chosen relative to the previous bond location. The complexity of this bristle model, which increases with N, makes it an inefficient model for use in simulations. Furthermore, lack of a damping factor may cause an oscillation under the sticking condition.

To make up for disadvantages of the bristle model, Haessig and Friedland also propose the "reset integrator" friction model. In this approach, instead of breaking

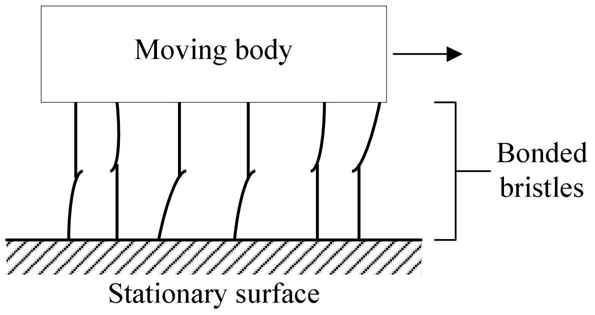


Figure 3.13: Illustration of the bristle model for friction

the bonds between bristles, the rate of increase in the bristle strain is kept constant after the rupture point is reached. This is accomplished by the introduction of an extra state *z* that determines the bristle strains:

$$\frac{dz}{dt} = \begin{cases} 0 & , \quad (v > 0 \ \ and \ \ z \geq z_0) \ or \ (v < 0 \ \ and \ \ z \leq -z_0) \\ v & , \quad else \end{cases} \tag{3.33}$$

The friction force is then given by

$$F = \left[1 + a(z)\right]\sigma_0(v)z + \sigma_1\frac{dz}{dt} \tag{3.34}$$

where $\sigma_0(v)$ is an arbitrary function of velocity that gives the friction when sliding (which may introduce the Stribeck effect), $\sigma_1 dz/dt$ is a damping term that is only active under sticking condition and $a(z)$ introduces the stiction effect that is given by

$$a(z) = \begin{cases} a & , |z| < z_0 \\ 0 & , \quad else \end{cases} \tag{3.35}$$

where z_0 is the maximum deflection before sticking condition terminates. This reset integrator model is more suitable for simulations than the original bristle model, but it is discontinuous in z.

Another detailed and accurate friction model based on the bristle interpretation of friction is the LuGre model given by Canudas de Wit el al. Here, the bristles deflect like springs under force. If the When the displacement is large enough, the bristles begin to slip. The equations for this model are given as follows:

$$\frac{dz}{dt} = v - \sigma_0 \frac{|v|}{g(v)} z \tag{3.36}$$

$$F = \sigma_0 z + \sigma_1(v)\frac{dz}{dt} + f(v) \tag{3.37}$$

where z is the average bristle deflection, σ_0 is the bristle stiffness, $\sigma_1(v)$ is the damping, $g(v)$ gives the Stribeck effect and $f(v)$ is the viscous friction. Common choices for damping, Stribeck effect and viscous friction functions are:

$$g(v) = \alpha_0 + \alpha_1 e^{-(v/v_0)^2} \tag{3.38}$$

$$f(v) = \alpha_2 v \tag{3.39}$$

$$\sigma_1(v) = \sigma_1 e^{-(v/v_d)^2} \tag{3.40}$$

For small displacements around zero velocity, the model acts similar to a spring-and-damper system. Linearizing (3.37) around z = 0 and v = 0:

$$F = \sigma_0 z + (\sigma_1 + \sigma_2)v \tag{3.41}$$

In (3.38), the sum α0 + α1 corresponds to the stiction force while α0 is the Coulomb friction force. Also, for constant-velocity steady state condition, the friction force simply becomes:

$$F = g(v)\operatorname{sgn}(v) + f(v) \tag{3.42}$$

3.4 Solver Techniques

Simulating dynamical systems requires the solution of governing state equations in order to obtain the state variables. Let a state x be defined with the ordinary differential equation (ODE)

$$\frac{dx}{dt} = x'(t) = f(t, x, u_1, \ldots, u_m) \tag{3.43}$$

where f is a function involving time t, the state x and inputs u_1 to u_m. Let x_0 be the initial value of this state, i.e.

$$x(t_0) = x_0 \tag{3.44}$$

The equations (3.43) and (3.44) then constitute an initial value problem (IVP). Although it may be possible to obtain analytic solutions to certain IVP's, there are many cases where this is difficult or impossible due to non-linearities or time-variant coefficients. Even when analytical solutions exist, the methods used to obtain them are only too many to be conveniently implemented on a computer-generated simulation process. In any case, the use of numerical integration methods (which only require the ODE itself) in order to obtain x becomes desirable in simulations.

Solver methods allow the approximation of the value of state x at time t, with the given initial condition x_0 at time t_0, based on the function f defining the ODE. Since these methods are computation-based, they provide solutions at certain time intervals. Specifically, the value of x is available at every time interval h (also called the "step size") after the initial condition, i.e. at $t = t_0$, t_0+h, t_0+2h, …, t_0+nh where n is an integer.

For convenience; the time, state and other values are represented in the indexed form (which also is used throughout this section)

$$t_n = t_0 + nh \tag{3.45}$$

$$x_n = x(t_n) \tag{3.46}$$

$$u_{m,n} = u_m\left(t_n\right) \tag{3.47}$$

where n is called the "time index". It should be noted that since numerical integration methods provide approximate values, it is mathematically more accurate to write the equation (3.46) with an approximately-equal sign ($\approx$). Still, the equal sign is used for the sake of convenience.

An advantage of these methods is that for any time t_{n+1}, the value of x_{n+1} only depends on up to p-many (depending on the specific method used) previous values of t, x and u. Therefore by keeping a p-long history of these variables, the calculations can be executed rather easily in simulations.

Some solver methods employ a variable step size h for calculation of x. The specific value of h is calculated such that the approximation error in each step is below a certain tolerance level, requiring an additional step to compute an estimate of the error. Variable-step methods allow for reducing the number of steps required for solution at the expense of accuracy. However, they are unsuitable for real-time simulations due to the difficulty in mapping the time step to a real-time clock. Furthermore, use of fixed-step methods is more desirable since the simulation itself operates on a fixed interval (i.e. the sampling interval of the controller). For these reasons, fixed-step methods are preferred throughout the thesis studies.

An important concern regarding the use of numerical integration methods is the stability of the solution. For the investigation of the stability, the term "stiff equations" are used. A stiff equation is a differential equation for which certain numerical integration methods turn out to be unstable unless the step size is smaller than a certain value. Stiffness generally arises when an equation contains changes in two very different time scales. For the definition of stiffness, let

$$x_{n+1} = P\left(h, f\left(t_n, x_n, u_{1,n}, \ldots, u_{m,n}\right)\right) \tag{3.48}$$

denote the application of a solver method *P* on the ODE given by (3.43). This ODE is then considered to be stiff if the Jacobian of *P* has at least one (complex) eigenvalue *m* that is outside the region of stability associated with the specific solver method selected. Since this eigenvalue is dependent on the step size, proper selection of *h* can remove the stiffness and allow for stable solution.

In the following subsections, various fixed-step solver methods are presented with brief discussions on their bases. Although the stability analysis is out of the scope of this thesis; it is evident that, in some cases, such a study might be needed in order to select a proper step size for successful simulation.

3.4.1 Euler's Method

The Euler method approximates the value of x_{n+1} by employing a finite difference approximation to x(t):

$$x'(t) \approx \frac{x(t+h) - x(t)}{h} \tag{3.49}$$

which, when rearranged, yields

$$x(t+h) \approx x(t) + hx'(t) \tag{3.50}$$

Using (3.43), the equation (3.50) can be written in time-indexed form as

$$x_{n+1} = x_n + h\, f\left(t_n, x_n, u_{1,n}, \ldots, u_{m,n}\right) \tag{3.51}$$

which is known as the "forward Euler method". This is an explicit method, requiring only already-known values for calculation.

3.4.2 Heun's Method

Also called the "modified Euler's method" or "explicit trapezoidal rule", Heun's method applies a predictor-corrector scheme, first calculating a rough approximation of x and then refining it. To calculate the predictor x^*_{n+1}, forward Euler's method is utilized:

$$x^*_{n+1} = x_n + h\, f\left(t_n, x_n, u_{1,n}, \ldots, u_{m,n}\right)$$

Then, the trapezoidal rule is applied as corrector to obtain x_{n+1}:

$$x_{n+1} = x_n + \frac{h}{2}\left[f\left(t_n, x_n, u_{1,n}, \ldots, u_{m,n}\right) + f\left(t_{n+1}, x^*_{n+1}, u_{1,n}, \ldots, u_{m,n}\right)\right] \tag{3.52}$$

3.4.3 Runge-Kutta Method

Runge-Kutta techniques, which are derived from the Euler method, involve the evaluation of the equation at multiple points within one integration step. The commonly used 4th order Runge-Kutta method, or “RK4”, is given by

$$x_{n+1} = x_n + \frac{1}{6}h\left(k_1 + 2k_2 + 2k_3 + k_4\right) \tag{3.53}$$

The variables k_1, k_2, k_3 and k_4 are given by

$$k_1 = f\left(t_n, x_n, u_1, \ldots, u_m\right) \tag{3.54}$$

$$k_2 = f\left(t_n + \tfrac{1}{2}h,\ x_n + \tfrac{1}{2}hk_1, u_1, \ldots, u_m\right) \tag{3.55}$$

$$k_3 = f\left(t_n + \tfrac{1}{2}h,\ x_n + \tfrac{1}{2}hk_2, u_1, \ldots, u_m\right) \tag{3.56}$$

$$k_4 = f\left(t_n + h,\ x_n + hk_3, u_1, \ldots, u_m\right) \tag{3.57}$$

Equation (3.53) is essentially the estimation of x_{n+1} using an estimated slope, which is a weighted average of different slopes (k_1, k_2, k_3 and k_4) within the time step.

3.4.4 Adams-Bashforth Method

Whereas the methods presented so far utilize the information from a single previous time step in order to calculate the value of x at the successive time step; the Adams-Bashforth method utilizes values from time steps up to four previous points, using a predictor-corrector scheme. The predictor x^*_{n+1} is given as

$$x^*_{n+1} = x_n + \frac{h}{24}\left(55f_n - 59f_{n-1} + 37f_{n-2} - 9f_{n-3}\right) \tag{3.58}$$

where $f_n = f(t_n, x_n, u_{1,n}, \ldots, u_{m,n})$. Then, x_{n+1} can be calculated using

$$x_{n+1} = x_n + \frac{h}{24}\left(9f^*_{n+1} + 19f_n - 5f_{n-1} + f_{n-2}\right) \tag{3.59}$$

where $f^*_{n+1} = f(t_{n+1}, x^*_{n+1}, u_{1,n}, \ldots, u_{m,n})$. It should be noted that since this method requires x_1, x_2 and x_3 in addition to x_0, it cannot self-start. Either these values either need to be provided as part of the initial conditions, or a method capable of self-starting (such as RK4) should be used until they are obtained.

3.5 Creating Discrete-Time Models of Dynamic Systems

In the modeling of dynamical systems, transfer functions are commonly used to express and analyze the relation between inputs and outputs of linear time-invariant systems, as well as many non-linear systems using appropriate linearization techniques. Transfer functions are obtained in the *s*-domain by the application of Laplace transformation on the governing differential equations and have the general form

$$G(s) = \frac{X(s)}{U(s)} = \frac{s^m\beta_m + \cdots + s^2\beta_2 + s^1\beta_1 + \beta_0}{s^n\alpha_n + \cdots + s^2\alpha_2 + s^1\alpha_1 + \alpha_0} \tag{3.60}$$

where *X* is the output and *U* is the input of the system. For purposes of discrete-time applications such as digital controllers or simulators, the discrete-time transfer function needs to be used. For this purpose, various transforms are used to convert an *s*-domain (continuous) transfer function into its *z*-domain (discrete) equivalent. This conversion maps the entire *s*-plane to the *z*-plane, where the imaginary axis is represented by a unit circle. All analysis and design tools regarding the performance and stability of a system are also applicable on this *z*-plane with appropriate mappings.

A common continuous-to-discrete transform is the Tustin method. This method uses a first order approximation of the natural logarithm function, performing the replacement

$$s = \frac{1}{T}\ln(z) \tag{3.61}$$

where *T* is the sampling time of the discrete-time model. Expanding ln(*z*),

$$s = \frac{2}{T}\left[\frac{z-1}{z+1} + \frac{1}{3}\left(\frac{z-1}{z+1}\right)^3 + \frac{1}{5}\left(\frac{z-1}{z+1}\right)^5 + \cdots\right] \tag{3.62}$$

approximating by taking only the first order term, the substitution for converting the transfer function to its discrete-time equivalent G_d is finally obtained:

$$s' = \frac{2}{T}\frac{z-1}{z+1} \tag{3.63}$$

$$G_d(Z) = G(s') \tag{3.64}$$

Let the general form of the discrete-time transfer function G_d be given as

$$G_d(z) = \frac{X(z)}{U(z)} = \frac{z^m\beta'_m + \cdots + z^2\beta'_2 + z^1\beta'_1 + \beta'_0}{z^n\alpha'_n + \cdots + z^2\alpha'_2 + z^1\alpha'_1 + \alpha'_0} \tag{3.65}$$

This function may be converted into a constant coefficient difference equation that can be employed to calculate the output *x* based on the input *u*. To obtain this equation, the numerator and denominator in (3.65) is divided by z^n:

$$\frac{X(z)}{U(z)} = \frac{z^{m-n}\beta'_m + \cdots + z^{2-n}\beta'_2 + z^{1-n}\beta'_1 + \beta'_0 z^{-n}}{\alpha'_n + \cdots + z^{2-n}\alpha'_2 + z^{1-n}\alpha'_1 + \alpha'_0 z^{-n}}$$

Replacing *z* with the unit delay *q*, *Y*(*z*) and *X*(*z*) can be converted into their time-domain forms *y*(*k*) and *x*(*k*) where *k* is the time index, giving the CCDE:

$$x(k)\left[\alpha'_n + \cdots + q^{1-n}\alpha'_1 + q^{-n}\alpha'_0\right] = u(k)\left[q^{m-n}\beta'_m + \cdots + q^{1-n}\beta'_1 + q^{-n}\beta'_0\right] \quad (3.66)$$

$$\begin{aligned} &\alpha'_n x(k) + \cdots + \alpha'_1 x(k-n+1) + \alpha'_0 x(k-n) \\ &\quad = \beta'_m u(k+m-n) + \cdots + \beta'_1 u(k+1-n) + \beta'_0 u(k-n) \end{aligned} \quad (3.67)$$

Provided that *n* > *m*, the CCDE can be easily solved for the value of *x* for any time index without requiring the future values of itself or of the input *u*. Arranging (3.67) in a more convenient form with this assumption, the CCDE finally becomes:

$$x(k) = a_1 x(k-1) + \cdots + a_n x(k-n) + b_1 u(k-1) + \cdots + b_m u(k-m) \quad (3.68)$$

where $a_i = \dfrac{\alpha'_{n-i}}{\alpha'_n}$ and $b_j = \dfrac{\beta'_{m-j}}{\alpha'_n}$.

3.6 Closure

This chapter provides a set of commonly used mathematical tools that can be used in modeling and simulating dynamical systems. Models for various dynamic system components, such as transmission elements and electric motors, were given. A study of the various friction models, which plays an important part in all systems, was made. Finally, techniques used in solving the ordinary differential equations governing a system in order to obtain its state variables were presented.

CHAPTER 4

HARDWARE-IN-THE-LOOP SIMULATION FOR EDUCATIONAL APPLICATONS

4.1 Introduction

Being the first approach to HILS used in the studies, this solution aims to provide a tool for reinforcing control systems education. It proposes the use of a personal computer for all the management and computation tasks associated with the simulation. While it serves as a means of exploring the capabilities of a purely PC-based simulation, its main goal is to provide a stand-alone and user-friendly educational tool intended for use in control systems related courses at universities.

In courses aimed at teaching discrete-time control systems, students are asked to design and test various controllers in order to reinforce the knowledge they receive. This makes it possible for the students to grasp every detail of the system they are controlling, helping them observe the effects of the methods and assumptions used during the design process. Without any doubt, the best way for such a reinforcement activity would be applying the given knowledge on real world systems.

This method, however, is not very practical. First of all, the system to be controlled may possess qualities that are unfit for being accommodated within a laboratory (such as size, noise, vibrations, toxic emissions etc.). It might be too costly for the university to purchase, set up and/or maintain. Furthermore, collecting information regarding the processes that occur inside the system is a difficult process involving proper fitting of the system with numerous sensors. Aside from these difficulties, it may be undesirable for the institution to make use of the same system over many years of education. In that case, need for acquisition of a new and suitable system (as well as proper salvaging, recycling or disposal of the old system) arises. Another inconvenience regarding this method is the impact of malfunctioning equipment on the students' laboratory works. Even with all the disadvantages taken care of, the experience the students are be able to gain will be very limited due to the fact that they need to use the system only within certain limited schedules.

In order to overcome the difficulties mentioned so far, this solution proposes the use of a "lab at home" approach in control systems education [90]. In this approach, instead of being dependent on a real system located at a laboratory, the students take advantage of a HILS application that imitates the system in question. They interface the controller they develop with this application and perform the simulation as if they were controlling the actual system itself. They are able to tweak with many system parameters such as disturbances, sensor noises, malfunctioning system components and so on. Thus, they are able to observe and work with almost all kinds of effects and problems that may arise in a real system. In addition to these capabilities, when a new system to work on is desired, it is sufficient simply to introduce its governing equations and parameters into the software package used.

4.2 Proposed System

A software package capable of simulating dynamic systems, named "Cadmus," is developed in the context of this educational package. The students use a development board provided to them to implement their controllers and couple it to their personal computers via a serial port. Software and documentation necessary to perform the communication between the PC and the board is provided to the students. The behavior of the system being controlled is realistically visualized on the software screens using 3-dimensional solid models, while the system's state variables are displayed on graphs and recorded on demand for later use. Outside effects on the system (such as disturbances, noise and loads) can also be manipulated using the software. By these means, students find the opportunity to test, tune and optimize their controller designs. The very same software is also used by the course instructor to evaluate the students' work.

Cadmus is developed using the C# programming language, dependent on *Microsoft .NET Framework*. It runs on *Windows XP* and *Windows Vista* operating systems. The operating system's graphical user interface is used for presenting user controls and application settings. Three dimensional solid modeling of the simulated system is done using the *Microsoft XNA Framework* [91], which is a cross-platform library intended for use in video games and provides methods for graphics, sound, input, networking and other gaming services. State variables, satellite and signal qualities and wind state are also overlaid on the 3-D view. Use of the mentioned frameworks and libraries has granted the application the ability to execute on almost any computer running the targeted operating systems.

Figure 4.1 shows the operational block diagram of Cadmus. The mathematical model and parameters belonging to the simulated system, user preferences and application settings form the *information* layer of the software. *Simulation* layer contains the numerical solver methods that will make use of the system's mathematical model to compute its state variables. Inputs to and outputs form the system are gathered from these state variables and incoming controller signals in order to be written into the *registers*; and presented for read and write operations over the *PC / controller interface*. The state variables are also sent to the *visualization* layer for three dimensional and graphical representations, as well as being recorded to the computer's storage devices on demand.

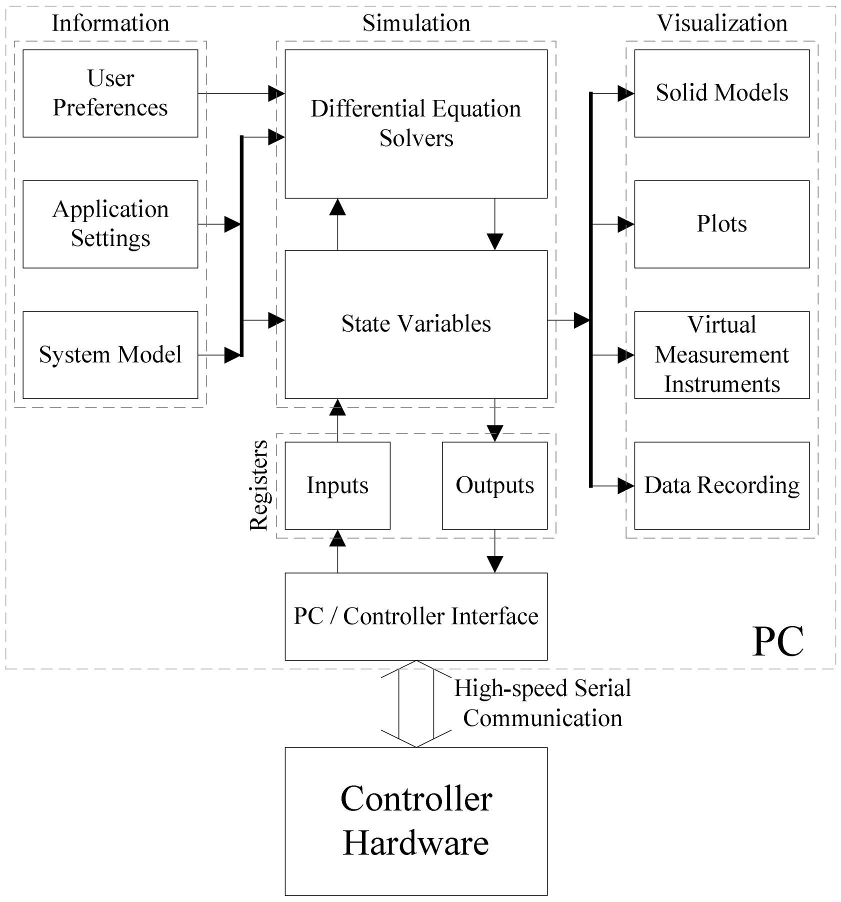


Figure 4.1: Operational block diagram for Cadmus software

When compared to other available software packages that can perform similar simulation tasks, the most prominent feature of the presented application is the three dimensional visualization of the simulated system. This allows the process of testing a controller to become much more than just an observation of the numerical values of the state variables (or their plots). The students find the opportunity to observe the system just like they would in an actual laboratory. Furthermore, it becomes possible to see the inner workings of sections that are impractical or impossible to expose (such as gearboxes, internal combustion engines, washing machines and so on) due to various reasons (lubrication, insulation, safety etc.) by means of using transparent models in visualization. Thanks to such features, the students' interests are kept awake while the simulation becomes a process that is easily understood with its causes and effects clearly visible.

As one expects, it is completely possible to also simulate the controller hardware (by use of programming or scripting languages) on the computer and get rid of the microprocessor used for implementation [92-93]. However, the system presented here purposely avoids such an abstraction and demands the use of an actual microprocessor. The motivation here is to introduce and familiarize the students with the electronic components on the board as well as programming and operational techniques that are unique to microprocessors. They are thus kept informed of various situations associated with the use of such devices. This prevents complete abstraction of the education from the real world, keeping it tangible.

4.3 Application

An application of the method presented thus far in this section has been done as the final project of the Computer Control of Machines course given at Mechanical Engineering Department of Middle East Technical University during 2007-2008 spring semester. This project asks the students to track a communications satellite in low orbit around the Earth with an antenna dish having two axes of motion: elevation and azimuth (Figure 4.2). The antenna is equipped with motors having integrated torque modulators, capable of applying the desired torque (within certain speed and power limitations–please refer to Section 5.3.2.1 for details). In addition, it is assumed that a navigational computer capable of calculating the position of the satellite with respect to the antenna (i.e. reference angles for elevation and azimuth), as well as a sensor measuring the strength of the received signal as long as the antenna is aligned with the satellite (Figure 4.3). The information generated by these subsystems is presented on 16-bit-wide registers to the PC interface. The students are expected to generate torque commands (again, 16-bit-wide) for the two motors on the axes. There is wind in the area where the antenna is located, blowing at speeds up to 30m/s (108km/h) and changing directions at random. Furthermore, there is a certain amount of backlash in the gears coupling the motors to the antenna, as well as Coulomb and viscous friction effects on the shafts. Under these circumstances, it is a serious control engineering problem to make the antenna track the communications satellite within tight tolerances during its two-minute flight on the visible sky. Such non-ideal conditions encourage the student to explore beyond the classical control algorithms and look for more advanced methods.

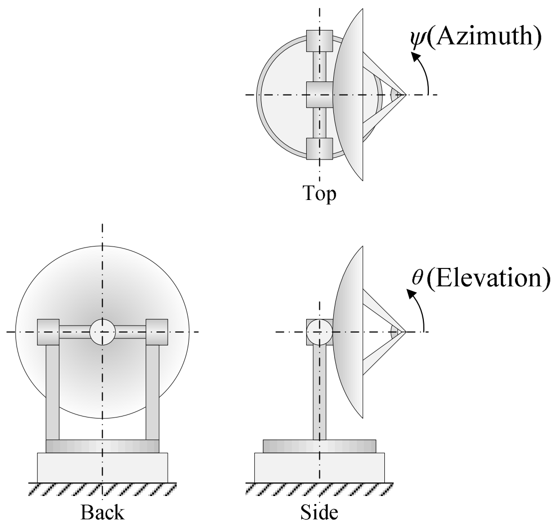


Figure 4.2: Representation of the antenna dish and the definition of angles

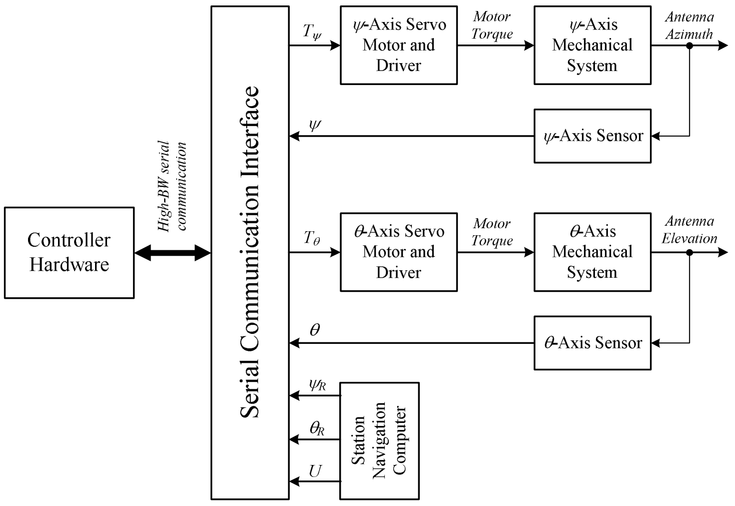


Figure 4.3: Functional block diagram of the Cadmus application project

4.3.1 Plant Model

As for the mathematical model of the simulated antenna system, since both axes are equipped with a gearbox (having a reduction ratio N_g) the below differential equations define the motion of each axis:

$$J_m \ddot{\theta}_m = T_\theta - T_g \cdot N_g^{-1} \tag{4.1}$$

$$J\ddot{\theta} + b\dot{\theta} + T_c \operatorname{sgn}(\dot{\theta}) = T_g - T_d \tag{4.2}$$

In these equations, J_m and J are the moments of inertia of the motor and mass reduced to axis shaft [kgm^2]; θ_m and θ are the angular positions of the motor and antenna [rad]; T_g is the torque output of the gearbox [Nm]; T_c is the Coulomb friction torque on the shaft [Nm]; T_d is the disturbance moment on the shaft due to wind force acting on antenna dish [Nm]; b is the viscous friction coefficient. Since backlash is assumed to exist, T_g is a function of the positional difference between the input and output shaft of the gearbox (see Section 3.2.1.1).
The strength of the signal received from the satellite in orbit is given by the exponential function

$$U(\psi,\theta,\psi_R,\theta_R) = 10 \cdot \exp\left[-\frac{(\psi_R-\psi)^2+(\theta_R-\theta)^2}{0.04^2}\right] \tag{4.3}$$

which exhibits a narrow peak of 10 Volts when the tracking error is zero.

4.3.2 Disturbance Model

The reaction of the antenna is modeled from the drag force applied on the dish, approximated by a hollow hemisphere. Consider the antenna dish, having frontal area A and looking at the azimuth and elevation angles ψ and θ, inside a uniform flow of air –wind– with density ρ (Figure 4.4). The velocity of the wind is given by the vector $\vec{V}$. The unit-length direction vector $\vec{u}$ for the dish can be obtained in terms of the unit vectors $\vec{i}$, $\vec{j}$ and $\vec{k}$ as:

$$\vec{u} = (\cos\psi)(\cos\theta)\vec{i} + (\sin\psi)(\cos\theta)\vec{j} + (\sin\theta)\vec{k} \tag{4.4}$$

The cosine of the angle γ between these vectors can be easily computed using the dot product

$$\cos(\gamma) = \vec{u} \cdot \frac{\vec{V}}{|\vec{V}|} \tag{4.5}$$

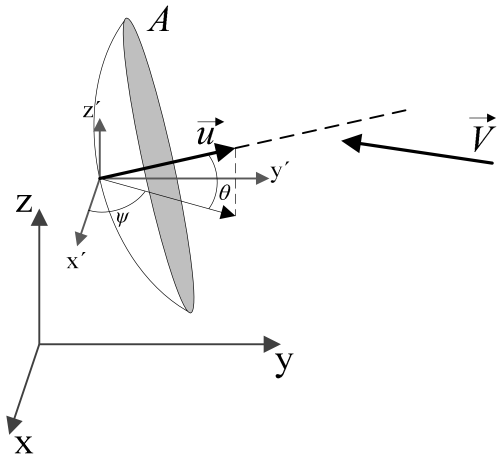


Figure 4.4: Antenna dish inside wind flow

The angle γ can be considered as the wind angle and used in calculating the drag area A_d as well as drag coefficient C_d. Figure 4.5 shows the orientation of the dish based on γ.

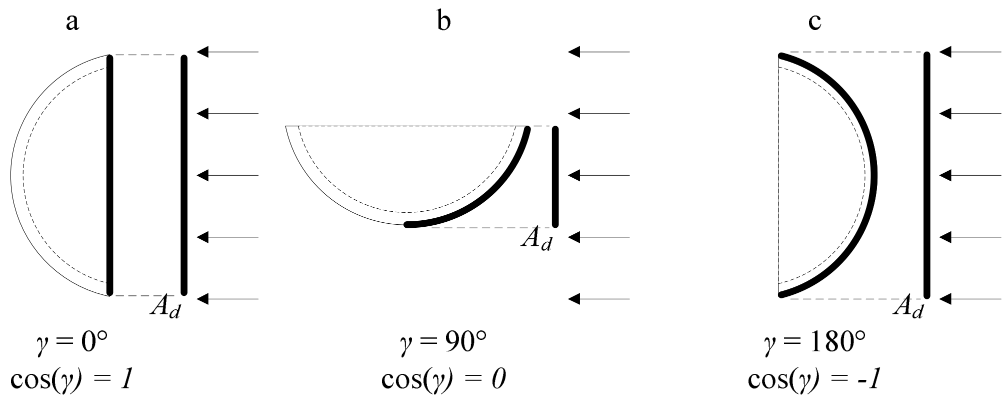


Figure 4.5: Wind angle of satellite dish

For the $\gamma = 0°$ and $\gamma = 180°$ cases, the drag area equals the frontal area, i.e. $A_d(0) = A_d(\pi) = A$, and for $\gamma = 90°$ the drag area equals the side area A_s of the dish and can be calculated as

$$A_s = A_d\left(\pi/2\right) = r_c^{\,2}\left[\alpha - \cos(\alpha)\sin(\alpha)\right] \tag{4.6}$$

where r_c is the radius of curvature of the dish and

$$\alpha = \sin^{-1}\left(\frac{r}{r_c}\right) \tag{4.7}$$

where r is the frontal radius. The drag area can then be approximated as a function of γ as

$$A_d(\gamma) = A_s + (A - A_s)\left|\cos\gamma\right| \tag{4.8}$$

As for the drag coefficient, Munson [94] gives the value of C_d for a parabolic dish as 1.42 for $\gamma = 0°$ and 0.95 for $\gamma = 180°$ while Peterka et al. [95] present consistent experimental work while giving a drag coefficient of 0.4 for γ = 90°. Based on their work, the drag coefficient can be expressed as a function of γ as

$$C_d(\gamma) = \begin{cases} 0.4 + 1.02\dfrac{1+\cos(2\gamma)}{2} & , \ \gamma \le \dfrac{\pi}{2} \\ 0.4 + 0.55\dfrac{1+\cos(2\gamma)}{2} & , \ \gamma \le \pi \end{cases} \tag{4.9}$$

The drag force on the antenna can now be calculated as

$$F_d(\gamma) = \frac{1}{2}\rho v^2 C_d(\gamma) A_d(\gamma) \tag{4.10}$$

Let r_m be the distance between the point P1, around which the dish rotates, and P2, the center point of the distributed wind force. Then the vector $r_m\vec{u}$ is the moment arm over which the disturbance torque is applied by the wind. The disturbance is then

$$\vec{T}_d(\gamma) = F_d(\gamma)\vec{u}_w \times r_m\vec{u} \tag{4.11}$$

And the torque on individual axes can be obtained as

$$T_{d,\psi}(\gamma) = \vec{T}_d(\gamma) \cdot \left[(\sin\psi)\vec{i} - (\cos\psi)\vec{j}\right] \tag{4.12}$$

$$T_{d,\theta}(\gamma) = \vec{T}_d(\gamma) \cdot \vec{k} \tag{4.13}$$

The antenna dish in the system is assumed to be 45cm in radius with a radius of curvature of 75cm. Using equations (4.6) to (4.10), the magnitude of the drag force can be calculated. Figure 4.6 shows the drag force on the dish with respect to the wind angle γ and the magnitude of $\vec{T}_d$ is shown in Figure 4.7.

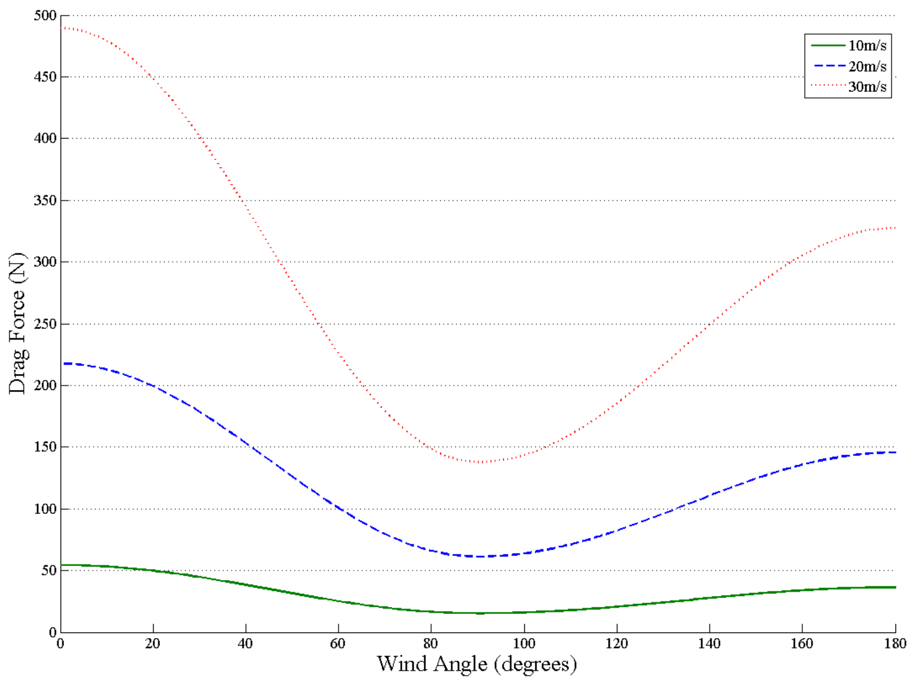


Figure 4.6: Drag force on antenna dish with respect to wind angle

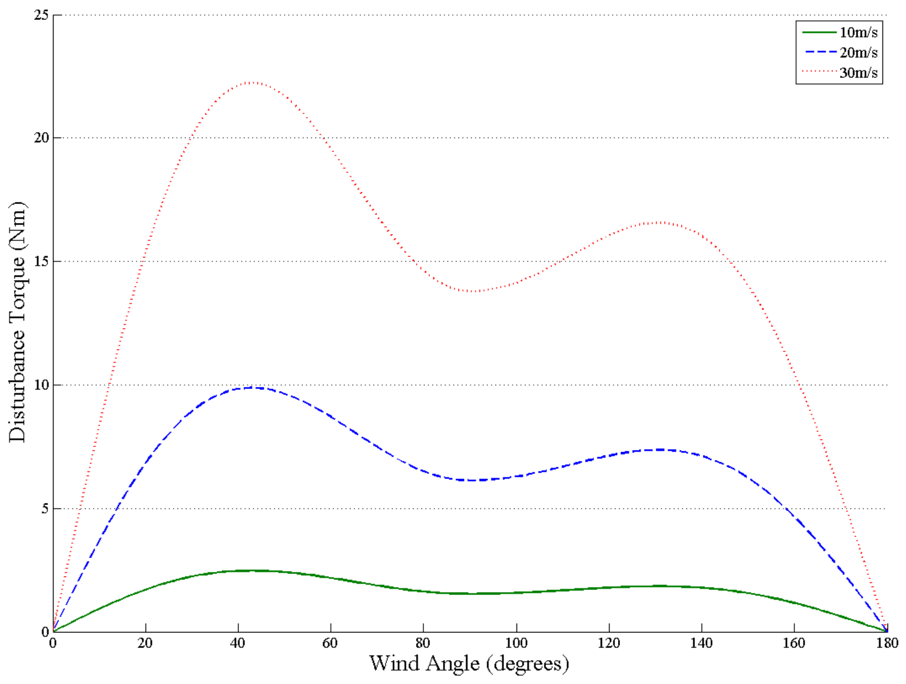


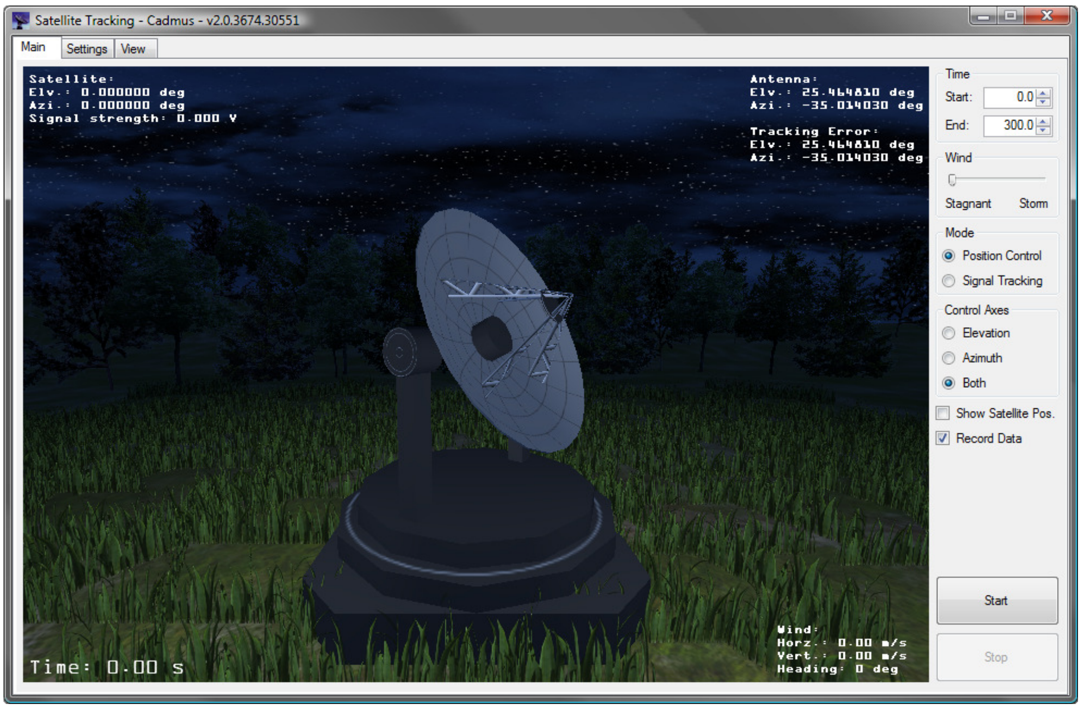
Figure 4.7: Disturbance torque on antenna dish with respect to wind angle

4.3.3 Software Details

To aid the students, the specific version of Cadmus developed for the project displays important information such as satellite position and disturbance torques in addition to the antenna's state variables. Along with these, data generated during simulation are both displayed on plots and recorded to a file. A three dimensional image of the antenna and its projections on its two axes are displayed on the user interface. Additional cursors on this display show the relative location of the satellite. A wind vane is rendered next to the antenna, hinting to the wind direction and speed.

The microprocessor provided the student is not a very capable one and it is unable to close the control loop in real time for high sampling rates. Rather, the write operations performed by the controller to the torque command registers cause the simulation time to advance exactly one sampling period. The computations are then paused and the software waits for the next set of commands. In the end, regardless of the actual time between write operations, the simulation computations are done assuming it is equal to the sampling time. This relieves the students from the burden of fitting the computations within a tight time frame, minimizing inconveniences caused by the development board and allowing them to focus on the control algorithm itself.

The software consists of three main screens. The *main* tab (Figure 4.8) displays position information on the antenna and satellite, wind status and simulation time in addition to the three dimensional visualization. In order to improve the image and keep the students' interest high, details irrelevant to the simulation but help in providing a realistic scene are also included: trees and grasses swaying according to the wind direction, clouds moving in the sky and a day/night cycle based on the computer clock. Controls provided in this tab allow the user to run the simulation for the desired time period or opt to work on a single axis instead of both.


Satellite Tracking - Cadmus - v2.0.3674.30551

```
Satellite:
Elv.: 0.000000 deg
Azi.: 0.000000 deg
Signal strength: 0.000 V

Antenna:
Elv.: 25.464810 deg
Azi.: -35.014030 deg

Tracking Error:
Elv.: 25.464810 deg
Azi.: -35.014030 deg

Wind:
Horz.: 0.00 m/s
Vert.: 0.00 m/s
Heading: 0 deg

Time: 0.00 s
```

Time — Start: 0.0 End: 300.0

Mode: [✓ Position Control] Signal Tracking

Control Axes: Elevation Azimuth [✓ Both]

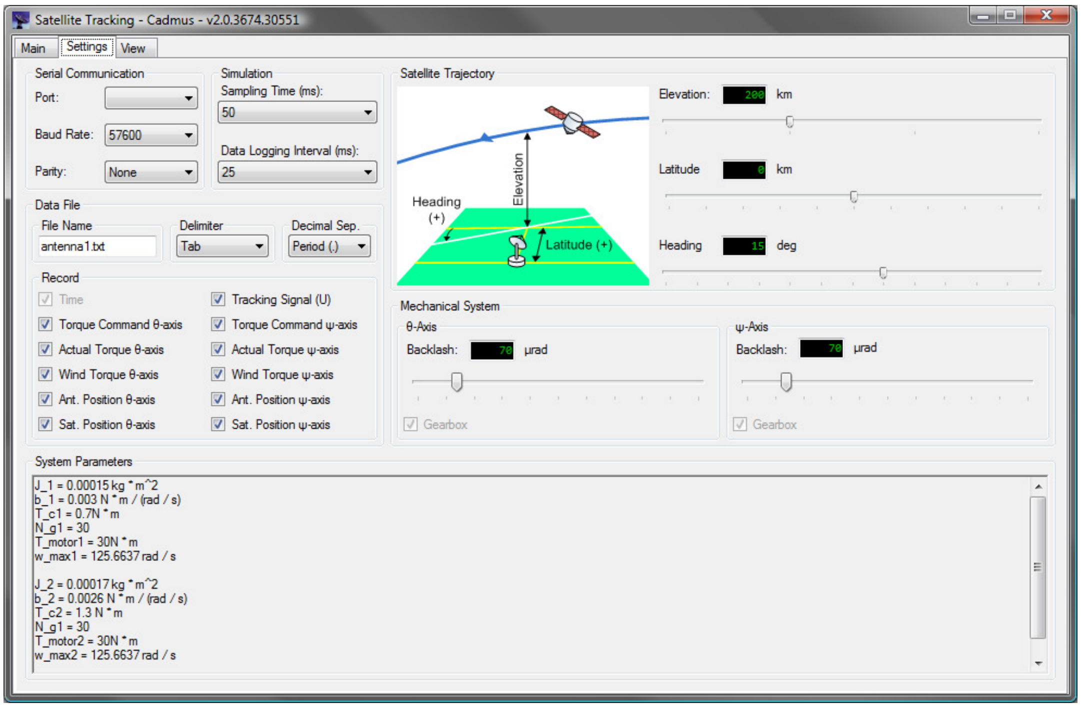
Show Satellite Pos. [ ]

Record Data [✓]

Figure 4.8: Main tab of Cadmus Software

The *settings* tab (Figure 4.9) presents certain parameters on RS-232 communication. The sampling time to be assumed during the simulation is entered in this tab. Settings regarding the recording of generated data also exist here. The user can select which of the many variables should be recorded; file name to be used and settings regarding tabulation and decimal separator for convenience when importing the data to other applications for analysis. Parameters defining the satellite orbit (elevation, angle and distance) can be changed, as well as the amount of backlash existent in the gearboxes.

The *view* tab (Figure 4.10) contains plots that visualize how the system's state variables change over time. There are two pairs of plots, one for each axis. First plot in each set shows the angular position of the relevant antenna axis and the relative satellite position. The second plot, on the other hand, shows the motor and disturbance (wind) torques applied on that axis. Also displayed in this tab is a plot of the received signal strength. During simulation, all plots are continuously updated and each of them can be zoomed and panned for viewing convenience.


Satellite Tracking - Cadmus - v2.0.3674.30551

Main | Settings | View

Serial Communication
Port:
Baud Rate: 57600
Parity: None

Simulation
Sampling Time (ms): 50
Data Logging Interval (ms): 25

Data File
File Name: antenna1.txt
Delimiter: Tab
Decimal Sep.: Period (.)

Record
- [x] Time
- [x] Torque Command θ-axis
- [x] Actual Torque θ-axis
- [x] Wind Torque θ-axis
- [x] Ant. Position θ-axis
- [x] Sat. Position θ-axis
- [x] Tracking Signal (U)
- [x] Torque Command ψ-axis
- [x] Actual Torque ψ-axis
- [x] Wind Torque ψ-axis
- [x] Ant. Position ψ-axis
- [x] Sat. Position ψ-axis

Satellite Trajectory
Elevation: 200 km
Latitude 0 km
Heading 15 deg

Mechanical System
θ-Axis
Backlash: 70 μrad
- [x] Gearbox

ψ-Axis
Backlash: 70 μrad
- [x] Gearbox

System Parameters

```
J_1 = 0.00015 kg * m^2
b_1 = 0.003 N * m / (rad / s)
T_c1 = 0.7N * m
N_g1 = 30
T_motor1 = 30N * m
w_max1 = 125.6637 rad / s

J_2 = 0.00017 kg * m^2
b_2 = 0.0026 N * m / (rad / s)
T_c2 = 1.3 N * m
N_g1 = 30
T_motor2 = 30N * m
w_max2 = 125.6637 rad / s
```

Figure 4.9: Settings tab of Cadmus software

Inside the software source code, all the components relating to the two axes of the antenna are collected under a single class. This class (with the addition of information from the registers and another class managing the wind behavior) refers to the solver class and obtains the angular positions and velocities of the two axes. This solver class is generically designed to accept any differential equation set introduced to it inside a certain template and employs a constant-step Runge-Kutta integration method of order 4 to solve these equations.

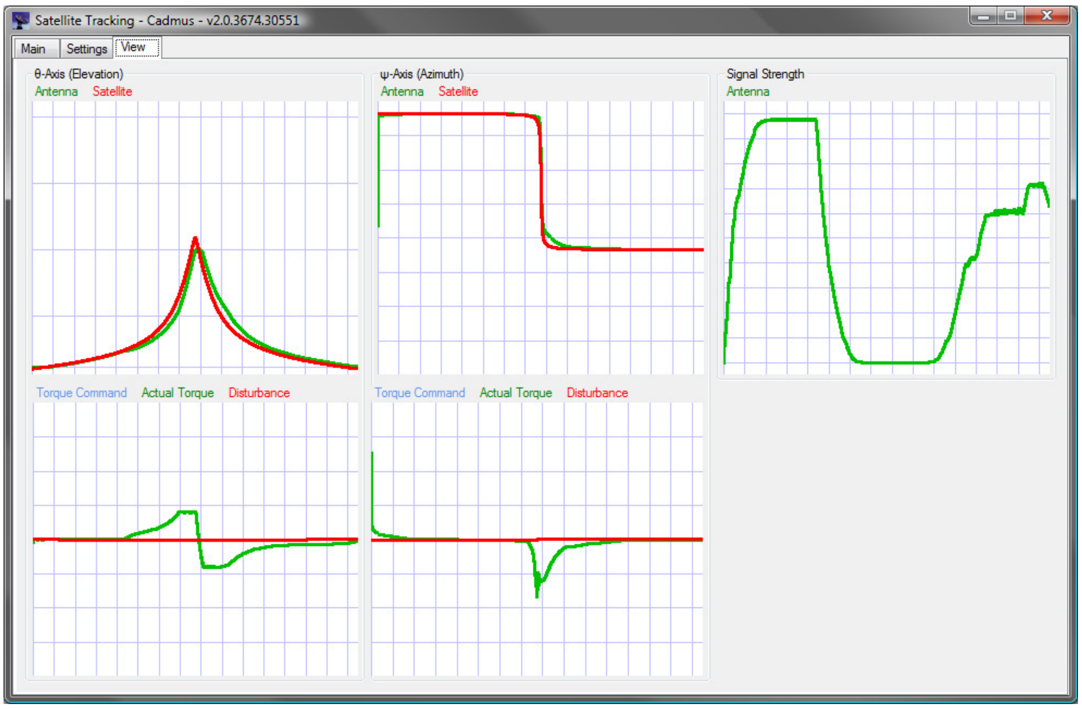


Figure 4.10: View tab of Cadmus software

The software executes multiple separate threads at run-time. The first thread performs the operations related with the *simulation* and *registers* layers mentioned in Section 5.2.2. When the simulation is started, the thread waits for the controller commands to arrive. When they are received; the simulation advances for one sampling period, the results are written into registers (to be queried by the controller) and the loop returns to command waiting state. The second thread executes the operations necessary for the three dimensional and projected visualizations of the antenna. A final thread is responsible for all other operations including the user interface and data recording. Thanks to this multi-threaded structure, the visualization operations (which are hardware-dependent and not easily done on every computer) are prevented from blocking the more crucial communication and simulation operations.

Another feature of the developed software is its ability to connect to the course web site to periodically check for updates to itself, whenever an internet connection is available. When a newer version is detected, the user is presented with the option to be directed to the web site for download and update news. This way, any changes made by the course instructor/assistants or the software developer can be delivered to the students as soon as possible.

4.3.4 Hardware Platform

A small, simple and low-cost development board designed for use with Microchip PIC18F4520 microcontroller unit; this board accommodates components useful for development and debugging of applications on such a microcontroller. The chip used on the board is a widely available general-purpose 8-bit microcontroller based on CMOS FLASH technology. Such a board meets the students' most basic needs, providing them a tool with which to implement controller algorithms while requiring only the simplest knowledge on electronics.

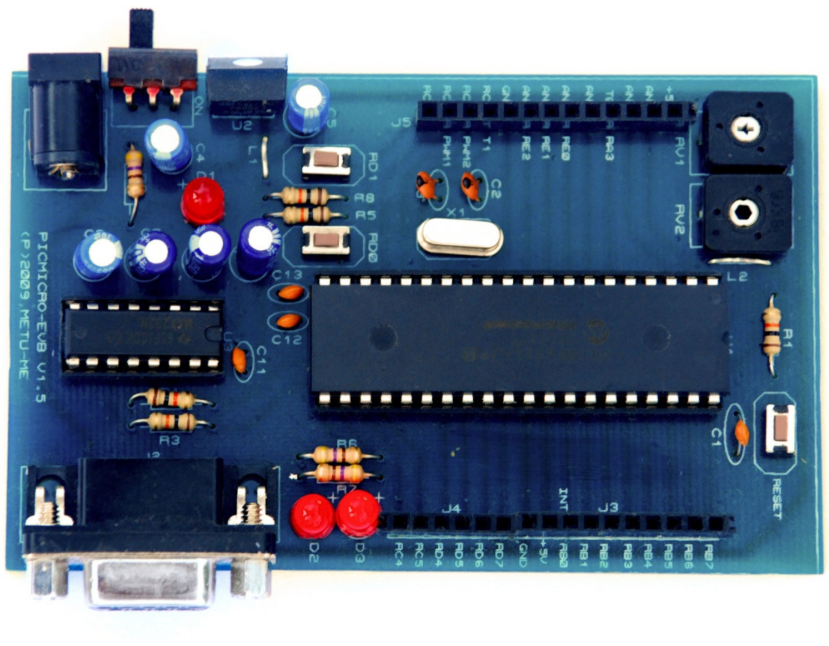


Figure 4.11: Microchip PIC development board

4.3.5 Interfacing

As stated in Section 4.3, transfer of 16-bit-wide data packets between the controller and simulator is required. Owing to the fixed packet size, it is possible to form a communication protocol that both performs fast and requires minimal memory footprint on the development board. The CCS C [96] source codes (which contain calls to built-in serial I/O routines) that employ this protocol are provided to the students and presented in APPENDIX B.

4.4 Results

For the solution of the control problem presented in the application, a variety of controllers were developed by the students who have taken the course in the project semester. These include PD, PI, PID and lead/lag controllers with command feed-forward, disturbance input decoupling, friction compensation and other similar components. One such controller developed for the project is a lead/lag controller designed via pole-zero cancelling root locus techniques by Mr. Mümin Özsipahi [97]. To improve friction compensation characteristics, a command feed-forward controller is also included. Finally, a disturbance observer enables high-performance rejection of the wind torque applied on the antenna dish. The overall architecture of this controller is shown in.

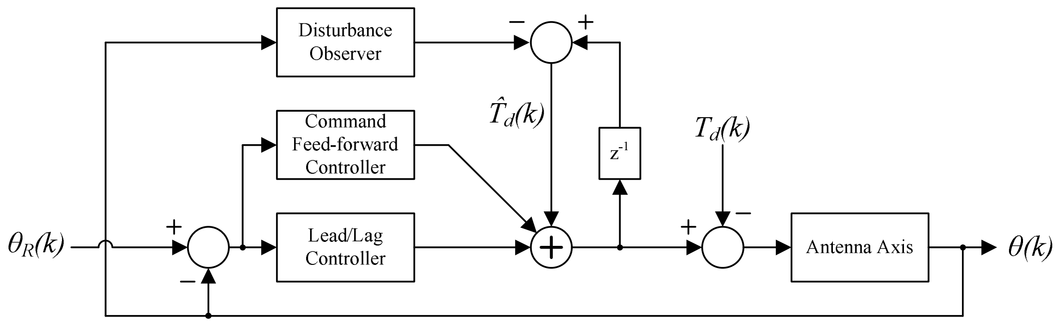


Figure 4.12: Lead/lag controller design for satellite tracking antenna

The performance evaluation of the controller is done on the Cadmus software. Multiple simulations are executed with different parameters. The magnitude and direction the wind acting on the dish changes at varying intervals. Figures 4.13 to 4.17 show antenna and satellite states and the received signal for a sample simulation run; demonstrating an exceptionally successful controller even under extreme disturbances. The strength of the received signal is almost always kept at maximum, only showing slight falls during initial application of high disturbance torques and a period of very fast change in the azimuth angle.

In order to evaluate the impact of the developed application on the learning process, the students who have taken the course have been asked to fill in a questionnaire regarding their experiences results for which are given in Table 4.1.

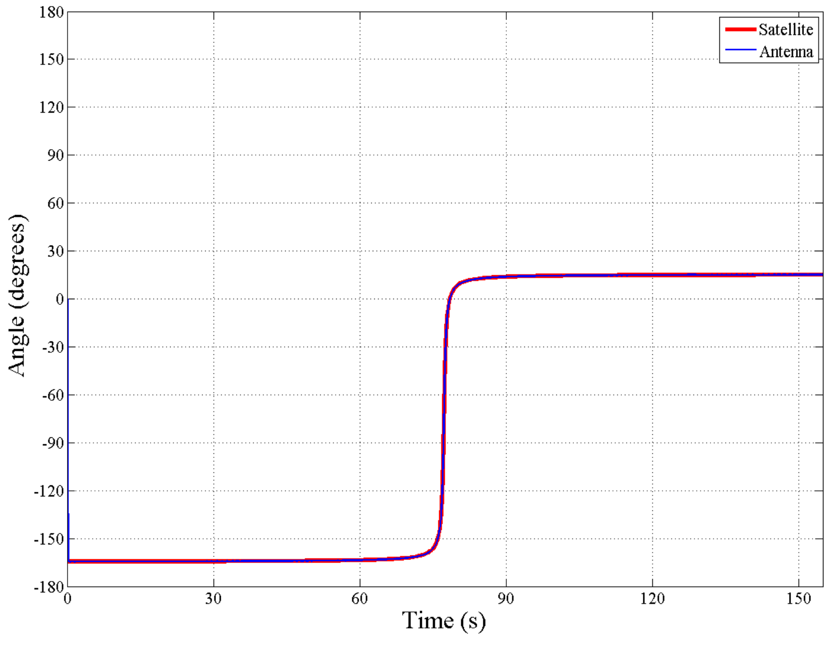


Figure 4.13: Satellite and antenna azimuth angles for sample simulation using lead/lag controller

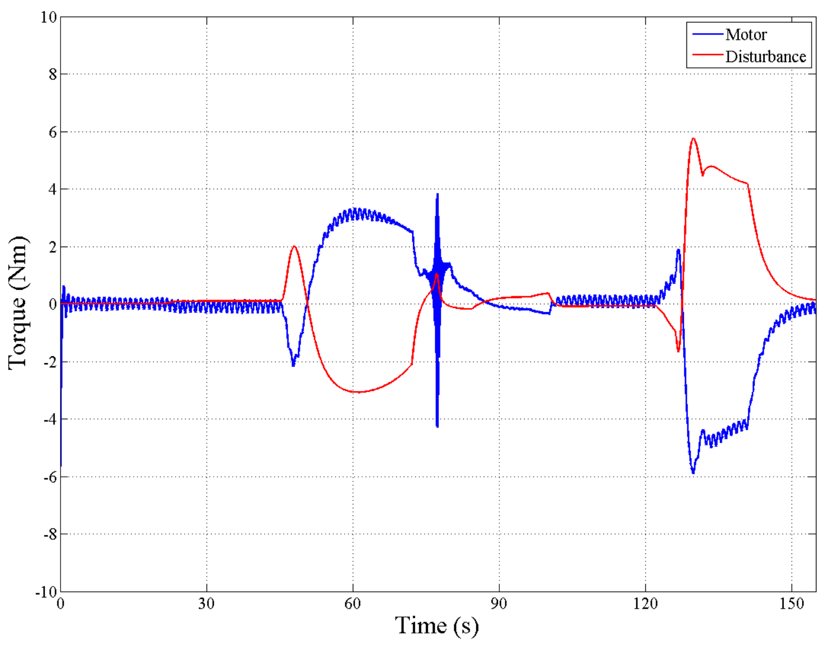


Figure 4.14: Motor and disturbance torques on azimuth axis for sample antenna simulation using lead/lag controller

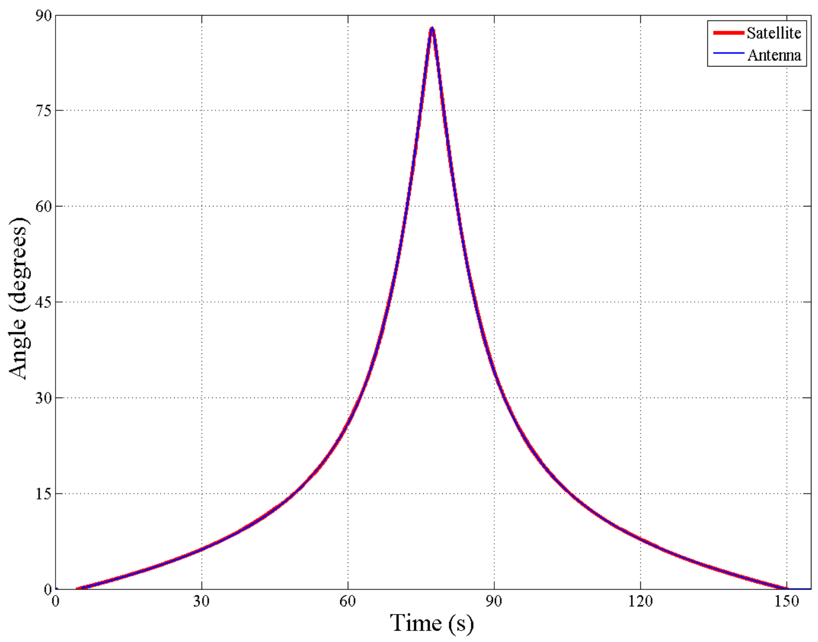


Figure 4.15: Satellite and antenna elevation angles for sample simulation using lead/lag controller

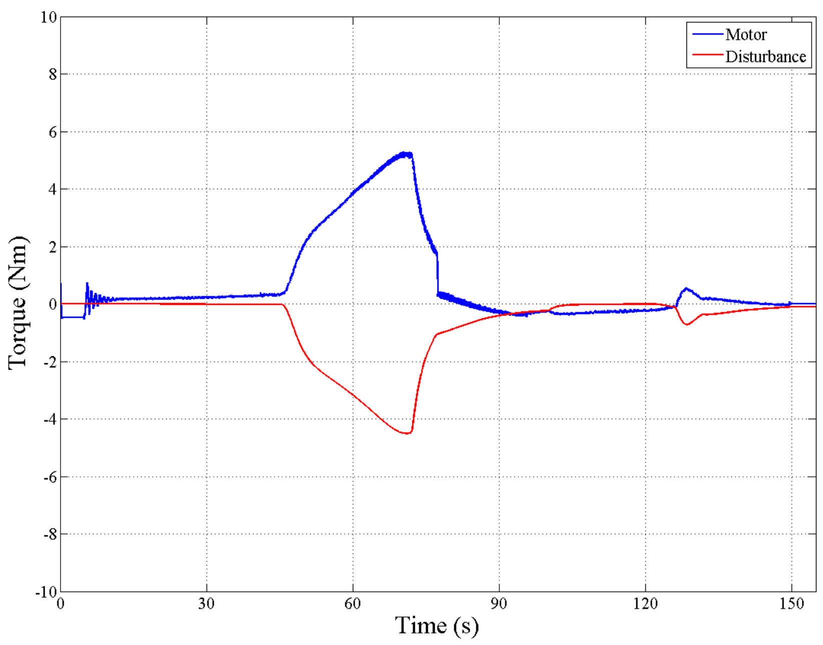


Figure 4.16: Motor and disturbance torques on elevation axis for sample antenna simulation using lead/lag controller

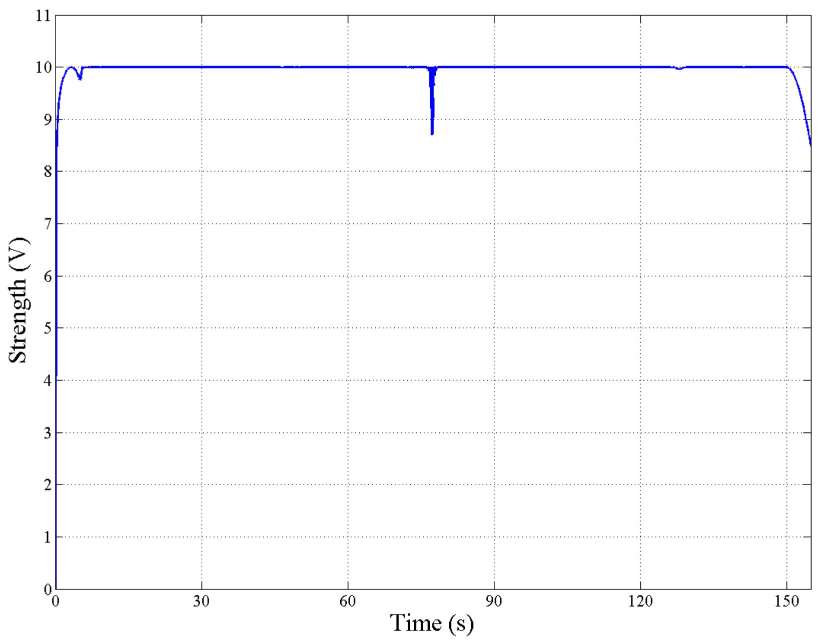


Figure 4.17: Signal strength for sample antenna simulation using lead/lag controller

Table 4.1: Questionnaire results for educational applications

		First application	Second application
Education program	Masters	64.3%	69.2%
	Doctorate	28.6%	23.1%
	Other	7.1%	7.7%
Department	Mechanical Engineering		92.3%
	Aerospace Engineering		7.7%
Average difficulty	Out of 10		8.08
Overall benefit of HILS	Out of 10		8.31
Effect of Cadmus on attitude towards the course	Positive	78.6%	69.2%
	Neutral	14.3%	30.8%
	Negative	7.1%	0.0%
Effect of Cadmus on final project period	More enjoyable	57.1%	69.2%
	Neutral	14.3%	15.4%
	Less enjoyable	28.6%	15.4%
Most helpful feature of Cadmus	3-D visualization		69.2%
	Recording of system states		7.7%
	Plotting of system states		15.4%
	Easy communication between controller and simulator		7.7%
Comparison of Cadmus with a similar application that can be developed using MATLAB	Prefer MATLAB	42.9%	15.4%
	No difference	21.4%	30.8%
	Prefer Cadmus	35.7%	53.8%

4.5 Closure

This chapter proposes an integrated HILS solution that is tailored for courses on control systems education. The overall concept of lab-at-home was explained and the software developed for realizing this concept was elaborated. The specifics on the actual application of the solution were presented along with the feedback received from the students who used the software.

For the purposes of the simple simulation with relatively lower sampling times used in the application, RS-232 communication protocol provides sufficient bandwidth. However, personal computer manufacturers have been abandoning the standard and almost none of the modern personal computers that target home and office users provide a serial port. Although USB to RS-232 converters are widely available in the market, student experiences show that product qualities vary wildly between many different brands. In most cases, these converters are reported to be the cause of communication delays that lengthen the simulation times to as long as 20 minutes, whereas use of certain brand converters or computers with a serial port allow the simulation to proceed virtually in real time. Therefore, instead of resorting to such unreliable converters, use of a development board that is capable of communicating over the more widely available USB ports is an improvement option.

In order to measure the simulation times, tests involving different state equations and combination of simulation elements are done on the developed software. Table 4.1 presents the results of these tests. These results show that the software is capable of closing one simulation loop within 181μs to 582μs. Based on these timings, simulating a system using 1ms sampling time almost in real-time is possible with the software. However, it should be noted that the introduction of higher degree state equations or those involving trigonometric, logarithmic or exponential functions are sure to have a large impact on these timings. At this point, programming and optimization methods that are native to the targeted operating environment need to be investigated.

Table 4.2: Cadmus simulation times

	Timings (milliseconds)			
Computer	A	B	C	D
1	0.006	0.058	0.096	0.181
2	0.022	0.193	0.347	0.582
A: No simulation (overhead due to communication and user interface tasks) **B:** Single-axis antenna simulation **C:** Two-axis antenna simulation **D:** Two-axis antenna, satellite and wind simulation				
PC 1: Intel Core 2 Duo 3.00GHz with 6MB cache, 2GB memory, NVIDIA GeForce 9600GT video card, Microsoft Windows Vista operating system **PC 2:** Intel Core Duo 2.00GHz with 2MB cache, 1GB memory, NVIDIA GeForce Go 7400 video card, Microsoft Windows XP operating system				

The student questionnaires show that the use of HILS in the course final project has an improvement on the overall attitude of the students toward the course. Understanding of the controller design processes were reinforced by their application on a working system instead of only using certain design and analysis tools. All students state that similar applications for all control-related courses would have a positive impact on their effectiveness. Specific to the Cadmus software, the students state that the 3-dimensional visualization helps them better grasp the behavior of the system under given inputs and conditions, as well as act as a source of enjoyment when they observe the results of their design. Although many of the students indicate that they could construct (albeit with difficulty) a HILS application using MATLAB/Simulink software package, they also express that they would rather prefer the Cadmus package. When the results from the two applications are compared, it is clearly seen that the improved and more bug-free software, as well as utilization of a more powerful development platform, increases both the effectiveness of the project and the enjoyment gained. As a result, the application successfully fulfills its goals.

Based on the studies presented in this chapter, a conference paper describing the lab-at-home concept and the Cadmus software package was [98]. It was submitted to Turkish National Committee of Automatic Control 2008 held in Istanbul, Turkey and was accepted for oral presentation.

CHAPTER 5

REAL-TIME HARDWARE-IN-THE-LOOP SIMULATION UTILIZING FPGA

5.1 Introduction

Aiming for a fast and highly integrated system, this solution utilizes an FPGA chip as the simulation's computing platform. This chip is used to instantiate solver for calculating the system states, as well as interface emulators necessary for communicating with the controller device. A number of peripheral units are connected to the FPGA: serial communication adapters, memory elements and storage devices. Additionally, a number of external interface cores (such as encoder pulse generators and PWM receivers) are instantiated on the FPGA chip in order to enable the simulation to communicate with external devices (e.g. the controller under test) or imitate peripherals (e.g. encoders or other devices). A PC connected to this platform serves as a means of managing simulation settings as well as monitoring system states during the simulation.

The solution uses CCDEs to calculate the simulated system's states based on their previous values in time and the inputs to the system. After the equations and parameters describing the system to be simulated are introduced to the microprocessor instance, the platform performs the simulation computations while communicating with the controller hardware. The system variables requested by the user are also sent to the user's PC during run time. If, however, the number of states requested is too many and the simulation step size is very small, this task requires a very large bandwidth. In such a case, the states are sent to the PC only at certain time steps. The rest of the states are saved on the storage elements on the platform, only to be dumped to the PC after the simulation is completed.

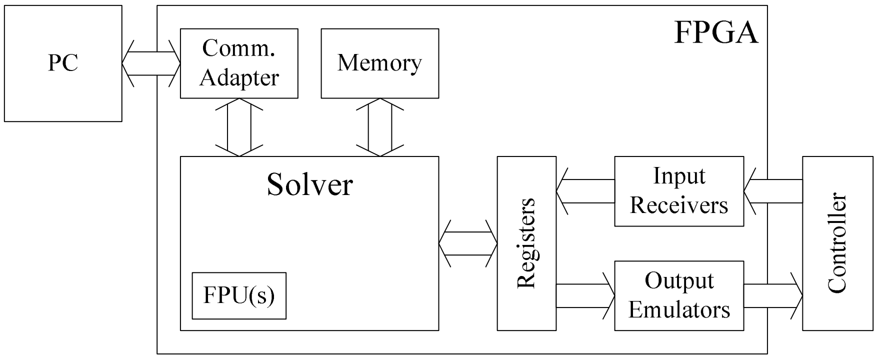


Figure 5.1: Hardware configuration for the FPGA solution

5.2 Application

For the realization of the FPGA-based simulator system, a simple HILS scenario is designed. An induction motor, which is used as the spindle drive of a turning center, is simulated in the application. The following sections explain the details of this system and its implementation.

5.2.1 Plant Model

The induction motor used in the system is assumed to be connected to a direct torque controller, which drives the induction motor. A timing belt connects the rotor shaft to the spindle shaft. A disturbance torque due to the cutting forces occurring during the machining process acts on the work piece. Figure 5.2 shows this spindle drive system.

In the figure, T^* represents the torque command, T_d is the disturbance torque, r_1 and r_2 are the pulley radii, J_1 and J_2 are the moments of inertia, b_1 and b_2 are the viscous friction coefficients, θ is the angular position of the spindle and e is the encoder signal.

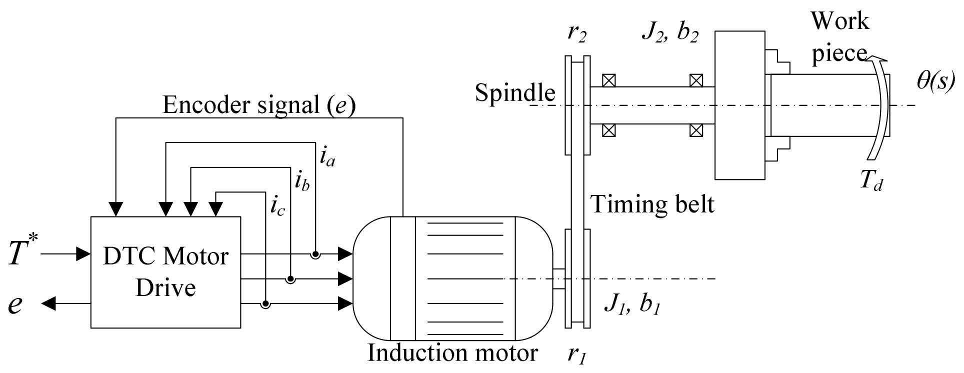


Figure 5.2: Spindle drive system

The DTC and induction motor in the system are assumed to be ideal. Therefore, these two can be modeled as a torque modulator with a torque capability curve, as explained in Section 3.2.2.

The transmission between the motor shaft and the spindle is assumed to be ideal, i.e. the timing belt perfectly transfers torque between the rotor and spindle shafts. It is then possible to model the combined rotor and spindle (and mounted work piece) loads as a single rotating mass. Then, the equivalent moment of inertia can be obtained by summing the rotor inertia with the spindle inertia, multiplied by the square of the transmission ratio. Likewise, the equivalent viscous friction is the sum of rotor friction with the spindle friction, multiplied by the transmission ratio.

The DTC in the simulated system is assumed to be receiving the torque command via a PWM resolver connected to a D/A converter. Since the simulation is performed digitally, instead of simulating a D/A converter, the inputs are fed into the PWM receiver and a 10-bit integer representing the torque command is obtained. The encoder emulator generates the encoder signals from the angular position of the spindle. These signals are counted by a quadrature counter and their difference is transmitted to the controller via another 10-bit PWM signal. The resulting simulated system model is given in Figure 5.3.

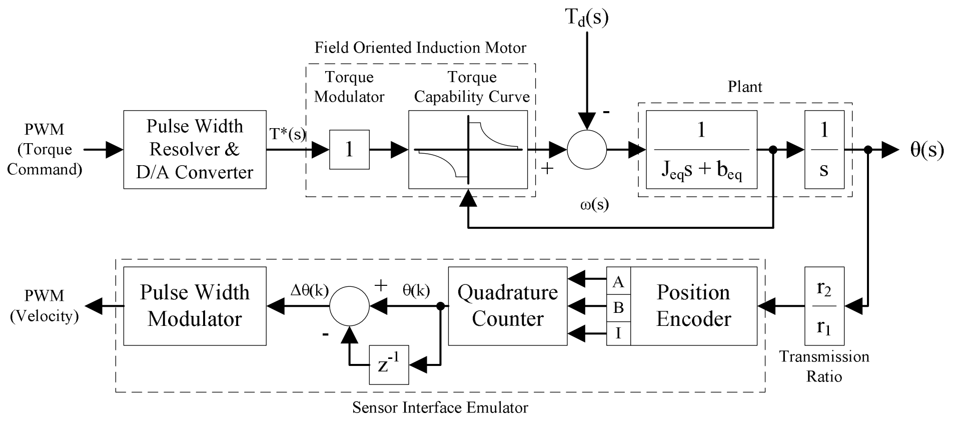


Figure 5.3: Block diagram of sample application system

For the purposes of the application, the rated motor torque is selected as 35Nm. The rated speed is 1500rpm while the maximum speed is 8000rpm, resulting in an approximate motor power of 5.5kW. The equivalent moment of inertia is taken as 0.07Nm^2, the equivalent viscous friction coefficient is taken 0.008Nms and the ratio of pulley radii is unity. Finally, using a sampling time of 1ms, the discrete-time transfer functions governing this system states as a function of the net torque (T_{net} = T_m - T_d) on the spindle shaft can be obtained as

$$\frac{\dot{\theta}(z)}{T_{net}(z)} = \frac{0.01428}{z - 0.9999} \tag{5.1}$$

$$\frac{\theta(z)}{T_{net}(z)} = \frac{7.143\times10^{-6}z + 7.142\times10^{-6}}{z^2 - 2z + 0.9999} \tag{5.2}$$

For a sampling time of 1kHz, the CCDEs of the system are then

$$\dot{\theta}(k) = 0.999\dot{\theta}(k-1) + 0.01428T_{net}(k-1) \tag{5.3}$$

$$\dot{\theta}(k) = 2\dot{\theta}(k-1) - 0.999\dot{\theta}(k-2) + 7.143\times10^{-6}T_{net}(k-1) + 7.142\times10^{-6}T_{net}(k-2) \tag{5.4}$$

The disturbance torque acting on the work piece (and spindle shaft) is taken as a repeating series of discrete pulses, imitating the effect of intermittent contact between the cutter and work piece during the machining process. The maximum magnitude of this disturbance is selected as 10Nm (Figure 5.4). In order to allow the motor to reach a high enough speed, the disturbance is introduced only after the simulation time reaches 8 seconds.

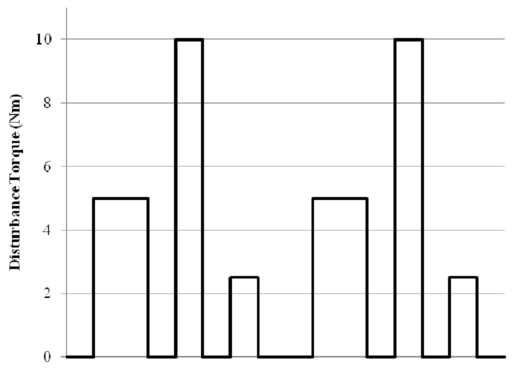


Figure 5.4: Form of the disturbance torque applied on the spindle shaft

5.2.2 Hardware Platform

For the implementation of the proposed solution, DE1, an FPGA development and education board manufactured by Terasic is used. This board carries a Cyclone II series FPGA chip from Altera. It provides a range of components and peripherals for easy introduction to the FPGA technology as well as to provide a handy tool for development, debugging and application.

5.2.3 Interface Emulators

Components that enable interfacing of the simulation with various controllers are presented in this section along with discussions on their workings.

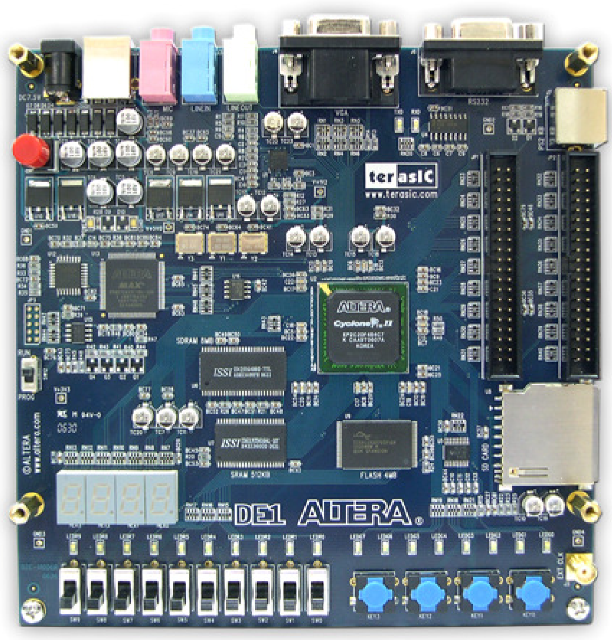


Figure 5.5: Terasic Altera DE1 Cyclone II FPGA Starter Kit [99]

5.2.3.1 Incremental Encoder emulator

Incremental rotary and linear encoders are quite commonly used in measuring the position of rotating or translating mechanical components. Incremental encoders use mechanical or optical sensors and a specially designed disc or linear scale to generate two square waveforms, with 90° phase difference. The change in these signals can be observed by resolvers to infer the magnitude and direction of the motion. The number of pulses output from one channel during one full revolution, or the “pulses per revolution”, determines the resolution of the encoder. The following figure illustrates the said waveforms.

The encoder emulator developed for simulation purposes aims to function by emulating both the edge transitions of the output signals and the timing of these edges. To do so, the emulator requires the value of the measured position, as well

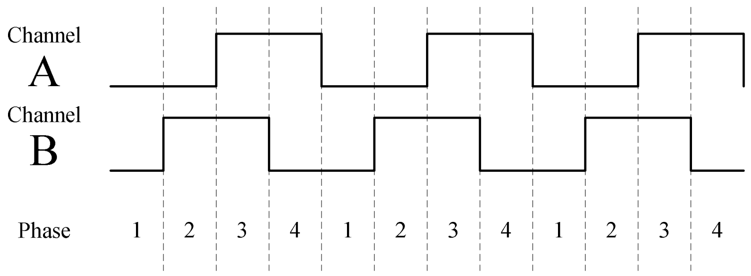


Figure 5.6: Incremental encoder output signals for clockwise rotation

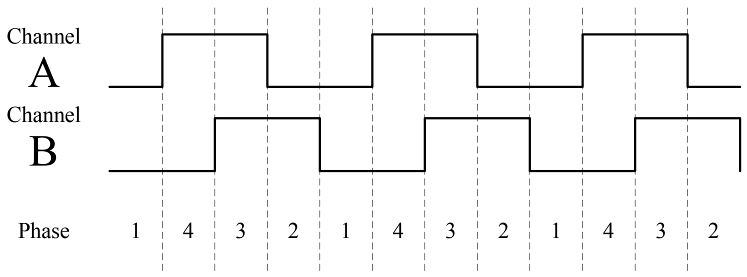


Figure 5.7: Incremental encoder output signals for counter-clockwise rotation

as the simulation sampling time and desired PPR value. Upon receiving command, the position is quantized to the given resolution. Comparing the quantized position with that from the previous sampling time, the number of pulses (or more precisely, edges) that should be output from the two channels are determined. These pulses are then sent via the output channels, evenly distributed within the sampling time. This ensures that the correct number of edges will be read in the correct duration by the resolver on the other end of the channels.

The implementation of the emulator contains an FPU for quantization, a configurable timer module and a counter for keeping track of the output signals. Taking advantage of the fact that the form of output signals can be separated to 4 unique phases, a two-bit counter can be used to provide these signals.

The fidelity of the emulator to real systems depends on the timing quality of the incoming commands. For optimal operation, the frequency of the received commands should be equal to the sampling time with zero deviation. Otherwise inconsistency in these timings may cause overlapping pulses to be skipped, causing error in the measured position.

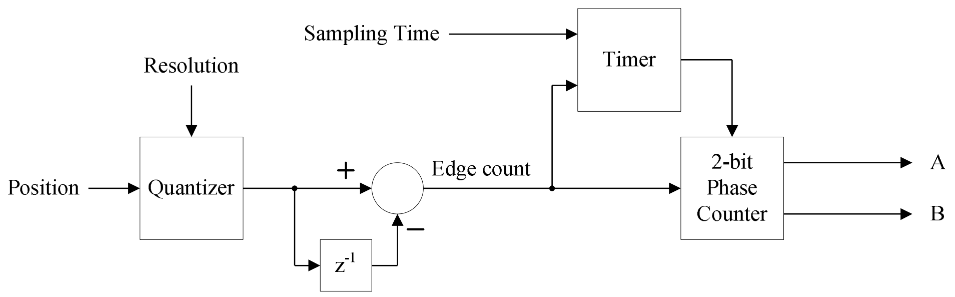


Figure 5.8: Incremental encoder emulator block diagram

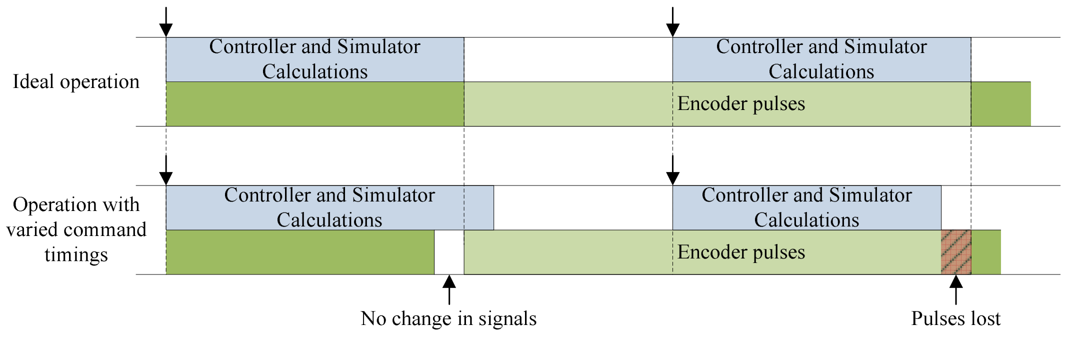

Ideal operation
Controller and Simulator Calculations
Controller and Simulator Calculations
Encoder pulses
Operation with varied command timings
Controller and Simulator Calculations
Controller and Simulator Calculations
Encoder pulses
No change in signals
Pulses lost

Figure 5.9: Illustration of timing in encoder emulator operation

5.2.3.2 PWM receiver

Pulse width modulation is a simple yet effective way for transferring data over a single line. Encoding data by modulating the width of a fixed-period pulse, i.e. changing the duty cycle, PWM signals are easy to both generate and resolve.
A simple PWM receiver implementation contains a counter that increments as long as the input signal is high and a timer that latches the counter value and resets it at the end of each sampling period.

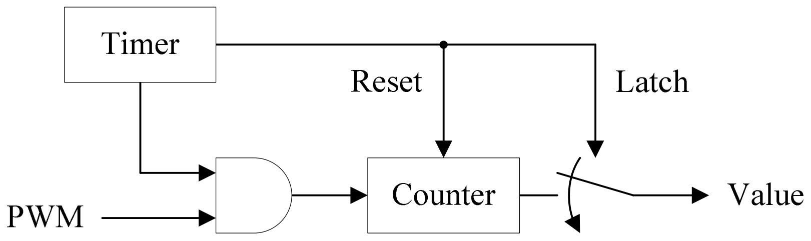


Figure 5.10: Block diagram of PWM receiver

To enable non-real-time simulation with controller as the clock source, a modified PWM scheme is also proposed. This scheme requires the use of a synchronization signal, which may be separate for all PWM channels or one global signal. When a falling edge on this synchronization signal is detected, the pulse beginning is assumed. After a fixed amount of time following this edge, i.e. the PWM period, counting is finished and the counter value is latched. The ratio of the high-time of the incoming signal to this period constitutes the duty cycle and thus the transmitted value. The end of the pulse period also marks the reception of the incoming command and a signal generated at this moment can be used as a trigger for the simulation. The time passing between the end of one pulse and the next synchronization is ignored.

The sole purpose of using a synchronization signal in the modified PWM receiver is to ensure proper simulation triggering. If the duty cycle of the incoming pulse can be guaranteed to be greater than 0% and less than 100%, its rising edge can be used for synchronization, eliminating the need for a separate signal.

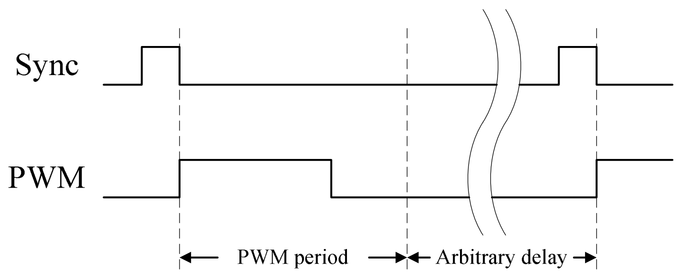


Figure 5.11: Modified PWM with synchronization signal

5.2.4 Embedded Microprocessor Implementation

The sample application is implemented on the Altera DE1 Development and Education board. This board accommodates the Altera Cyclone II 2C20 FPGA along with an 8MB SDRAM chip, USB Blaster serial communication adapter, a 50MHz oscillator and many other useful peripherals. On the FPGA chip, an instance of the Nios II 32-bit Embedded Processor, which is designed by Altera solely for FPGA implementation, is implemented. This processor design includes an FPU implementation, and can be connected to modular interfaces to a variety of peripheral units which include those on the DE1 board. The Nios II processor and the interfaces needed to utilize the aforementioned peripherals are instantiated and connected together using the development tools provided by Altera, and an SOPC is obtained. This system is coupled with the encoder generator and PWM transmitter and receiver components, and downloaded onto the FPGA chip.

A C program that will perform the simulation calculations of the explained system is written and compiled for the Nios II processor. The program consists of a main loop that repeatedly performs the communication and computation tasks. Instead of a continuously running loop, the program waits for a torque command to arrive from the controller. When a command is received, an interrupt is raised in the processor. The incoming command is read from the digital I/O interface, and the simulation computations for a single sampling period are performed. The shaft position, which is used to generate the encoder signal, is outputted from the processor via the digital I/O interface to the relevant register. All system states, as well as the torque command and net torque, are recorded in arrays stored inside the SDRAM chip. Finally, when the simulation is completed, these records are dumped to the user's PC via the USB connection.

5.2.5 Parallel FPU Implementation

For the purpose of investigating the capabilities of the FPGA platform, a partial implementation of the application was also done by instantiating multiple floating point operation units directly on the chip. This implementation aims to exploit the full parallelism potential of the said platform. For every arithmetic operation in the CCDEs (5.3) and (5.4) governing the system, FPUs are placed in a manner suitable for parallel processing, forming a binary-tree like structure (Figure 5.12). In addition to these, FIFO type memory modules that expose all their contents are placed in order to hold previous time values of system states. A management module is also placed to operate the FPUs.

The solutions of the CCDEs are initiated by the simultaneous operation of the floating point multiplier blocks by the management module. The results of these multiplications are fed into the addition blocks, and these are operated in turn until the results of the equations are obtained. These results are finally fed into the relevant FIFO buffers to be used during the next computation loop.

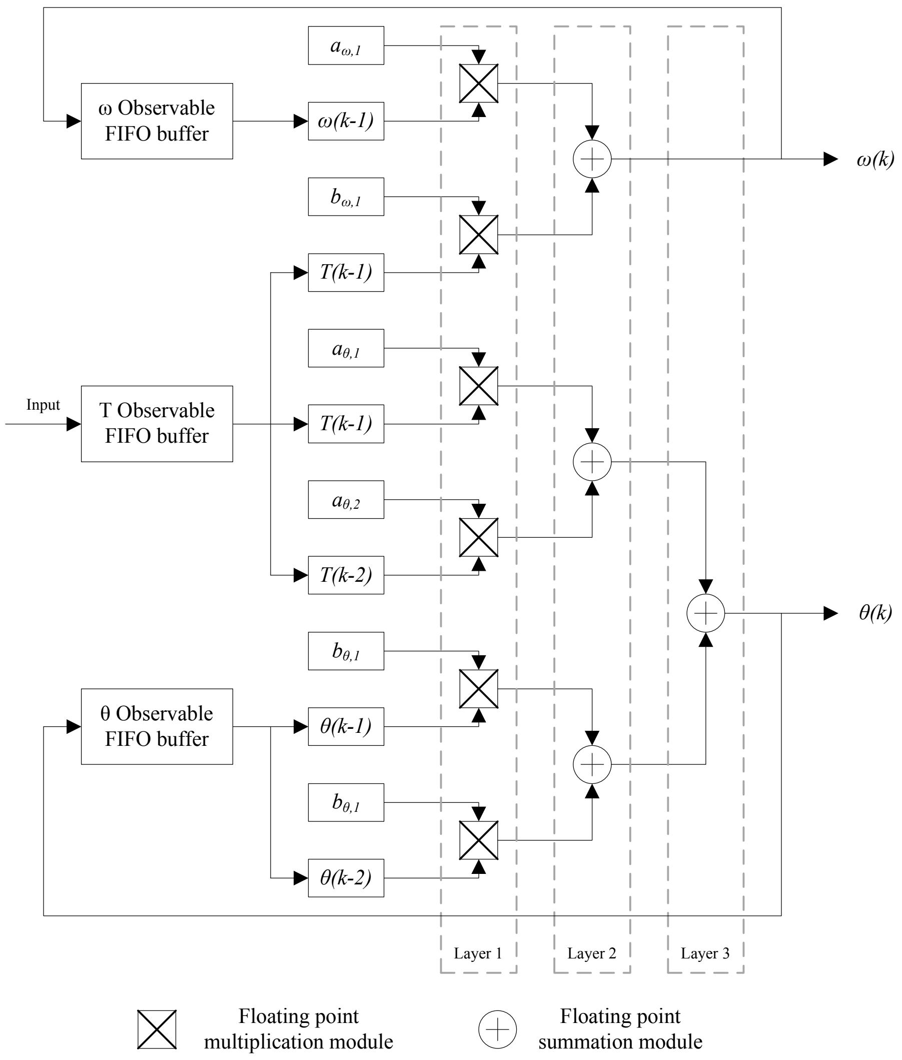


Figure 5.12: Block diagram for parallel-FPU CCDE solver

5.3 Results

The offline testing of the application is performed by applying constant torque command with 30Nm magnitude on the system. The results obtained are compared against solutions performed on PC using the MATLAB/Simulink software package. Two different implementations are tested, using single- and double-precision floating point representations. The performance of the simulator is also evaluated in terms of computation time by utilizing the performance counter component of the Nios II processor. Table 5.1 presents the test results while the resource usage of the application is given in Table 5.2.

Table 5.1: Offline simulation test results

		Single-precision implementation	Double-precision implementation
Difference from Simulink in ω	RMS	410.1μrad/s	409.2μrad/s
	Mean	51.5mrad/s	51.3mrad/s
	Maximum	240.2mrad/s	240.2mrad/s
Difference from Simulink in θ *	RMS	3.7μrad	3.7μrad
	Mean	466.9mrad	470.3mrad
	Maximum	869.2mrad	881.4mrad
Computation Time	RMS	13.3μs	111.7μs
	Mean	13.2μs	111.5μs
	Maximum	26.8μs	140.4μs

* After 20 seconds of simulation.

Table 5.2: Sample application resource usage

Total logic elements	10,737 (57%)
Total combinatorial functions	9,121 (49%)
Dedicated logic registers	6,447 (34%)
Total memory bits	66,040 (28%)
Embedded multiplier 9-bit elements	11 (21%)
Program size in SDRAM memory**	52kB (0.006%)

* Percentages are based on resources on the Altera DE1 board.
** Excluding stack and heap memories.

The numerical results of the simulation appear to have an acceptable amount difference from those obtained using Simulink, due to the difference between the employed solution techniques. It should be noted here that the error in angular position is the integral of the error in speed and will vary with the simulation duration. Again in terms of numerical results, there appears to be no significant difference between single- and double-precision floating point representation techniques. The single-precision solution times, however, are much lower than double-precision solution times. The single-precision implementation is therefore an adequate solution as far as the sample application is concerned.

In order to further explore the performance of the sample application, the simulation program is modified to solve for CCDEs of varying degrees as well as more than one equation at a time, and the computation times are measured. Results are given in Figure 5.13 and Figure 5.14.

The parallel-FPU implementation is capable of performing a floating point operation every 4 clock cycles [100]. In the binary-tree arrangement of this implementation, the solution time can be calculated as

$$n_c = 4 \times \left(\left\lceil \log_2(n_t) \right\rceil + 1 \right) \tag{5.5}$$

where n_t is the number of terms added together in the CCDE and n_c is the number clock cycles required to obtain the solution. For this implementation, the solution times using a 50MHz clock signal are given in Figure 5.15, and Figure 5.16 shows the approximate resource usage.

A final comparison is made with a numerical method approach, the Runge-Kutta algorithm of order 4. The method is implemented on the Nios II processor, only modifying the solution part of the simulation program. The state space equations governing the system are obtained and also coded into the solver, and numerical integration is performed. Two tests are made, one with a single step per loop (1ms step size) and one with 10 steps per loop (0.1ms step size). The computation times are presented in Table 5.3, while the resource usage is identical to the originally proposed solution.

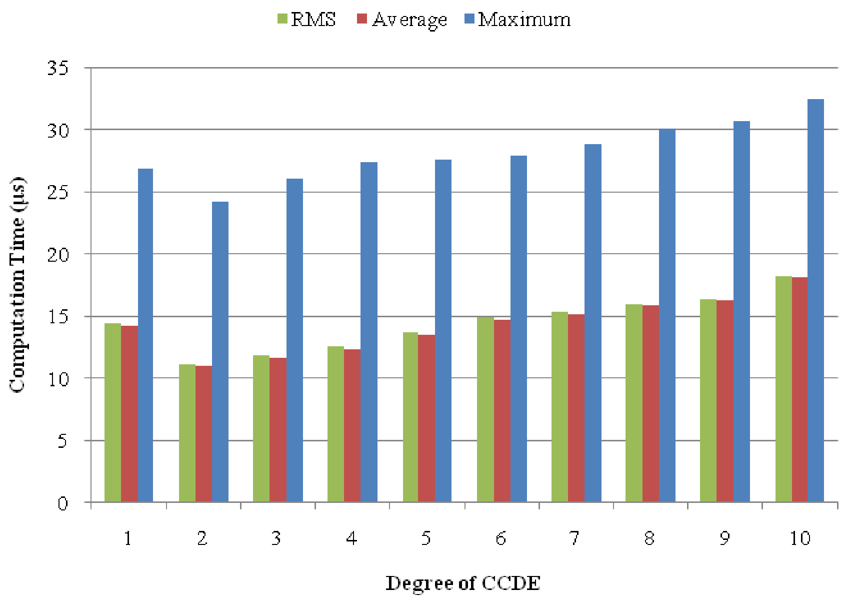


Figure 5.13: Computation time by CCDE degree, one equation per loop

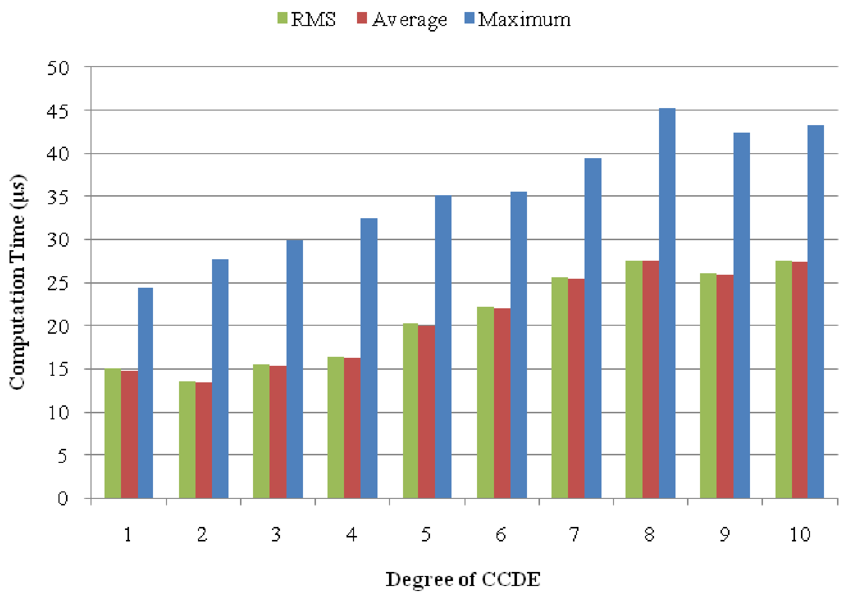


Figure 5.14: Computation time by CCDE degree, two equations per loop

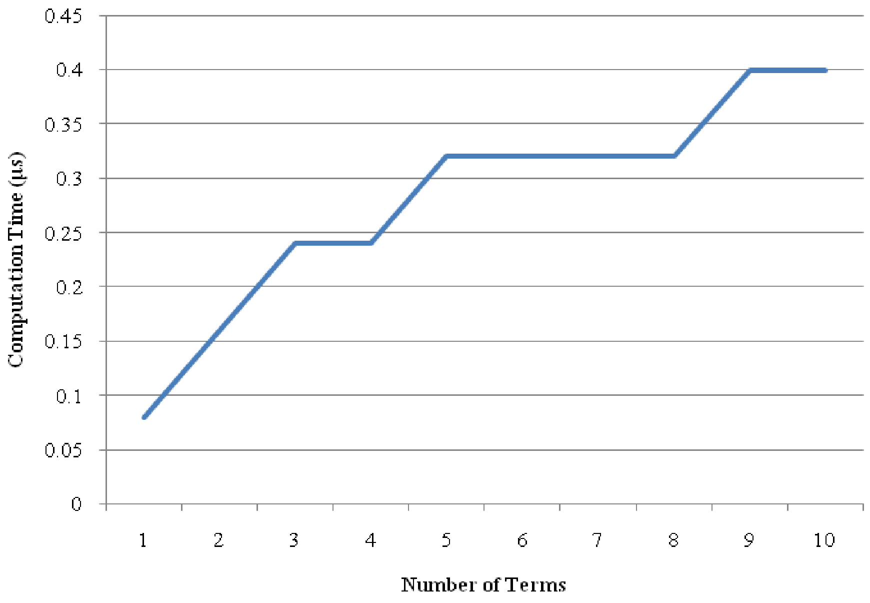


Figure 5.15: Parallel-FPU implementation computation time by CCDE degree

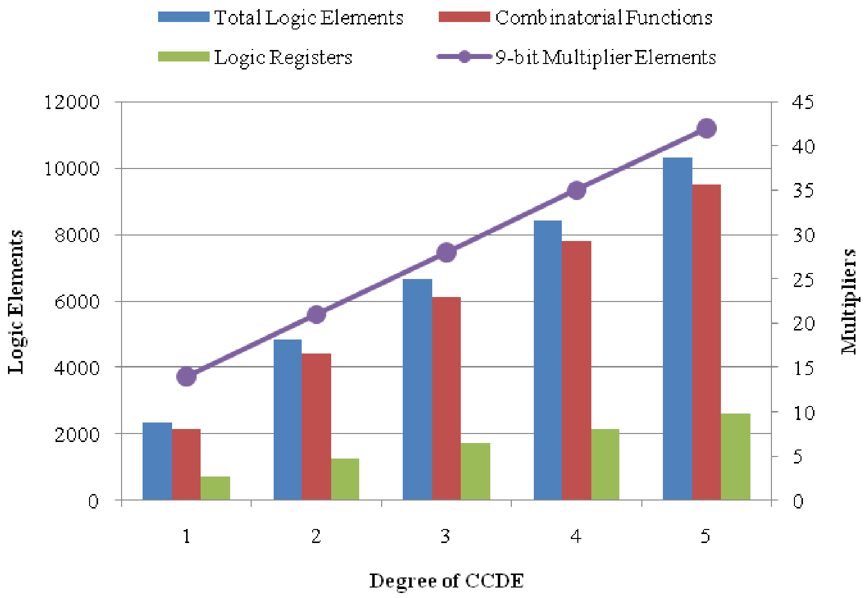


Figure 5.16: Parallel-FPU implementation resource usage by CCDE degree

Table 5.3: 4th Order Runge-Kutta method computation times

	1 step per loop	10 steps per loop
RMS	19.1μs	63.5μs
Mean	19.0μs	63.4μs
Maximum	36.0μs	70.3μs

Once the application is observed to be operational, it is used in evaluating the performance of various FPGA-based controllers that employ different control algorithms. These controllers use hysteresis, fuzzy logic, PID and sliding mode control schemes to keep the angular velocity of the spindle on a reference input that linearly increases up to 1200rpm and remains constant afterwards. Figures 5.17 and 5.18 show the performance of these controllers.

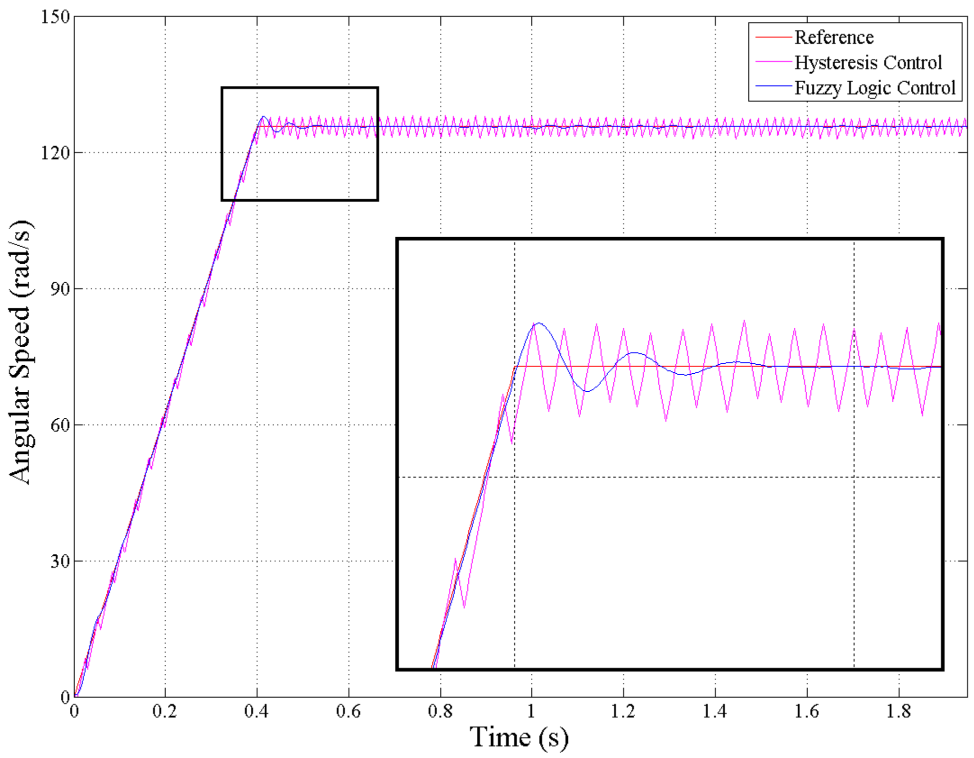


Figure 5.17: Hysteresis and fuzzy controller performances for spindle drive

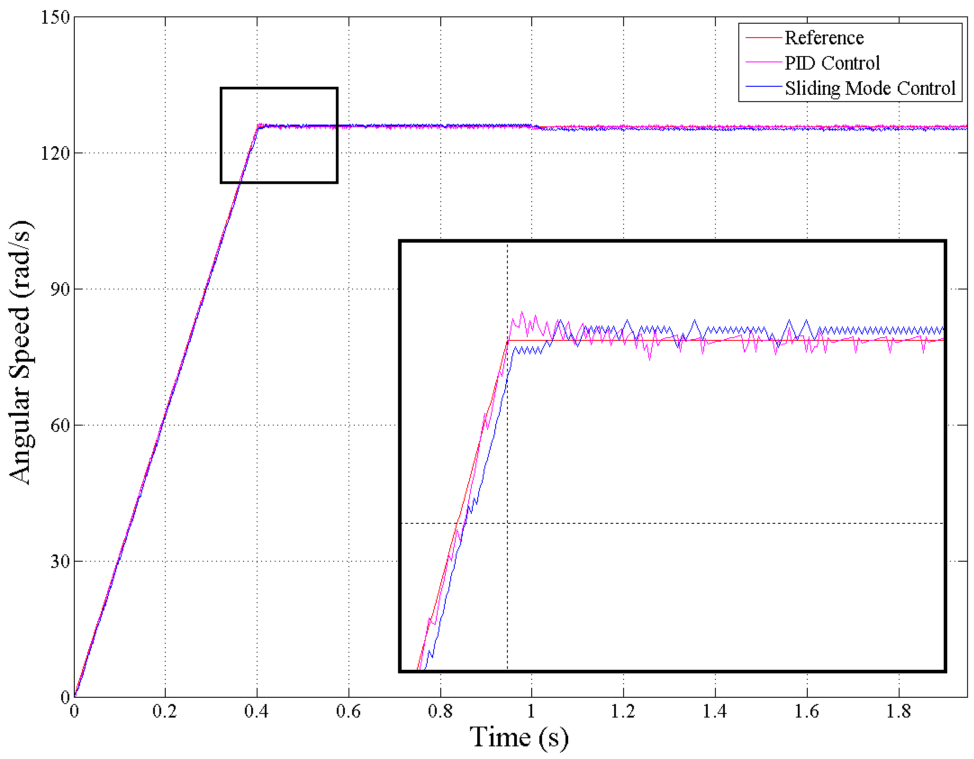


Figure 5.18: PID and sliding mode controller performances for spindle drive

5.4 Closure

This chapter builds up a general-purpose HILS solution that may be used as a basis for developing other, more complex simulations. The proof of operation, as well as the advantages and disadvantages of the proposed system are investigated using a simple application.

The spindle drive is a good demonstration of the computational capabilities of the proposed solution. The CCDE solution times show that it is possible to successfully simulate systems of order up to 10 using sampling times as high as 5 kHz.

Although implementing a direct hardware solution method makes it possible to perform the computations at much higher speeds, the rapidly increasing resource usage limits the capability of this approach. Moreover, introduction of non-arithmetical operations, such as conditional statements, requires redundant resource usage in order to maintain the desired computational speed, further limiting the method's capabilities.

While solving CCDEs to obtain system states is quite fast, it has the limitation of only being able to represent linear systems. When non-linear elements exist in the system to be simulated, linearization techniques become necessary to obtain a CCDE, but the accuracy of the model is greatly reduced. In such cases, it is possible to resort to numerical methods such as the RK4. It is also shown that the proposed solution can be easily modified to use this approach, still being able to perform simulations with sampling times up to 2 kHz successfully.

Based on the studies presented in this chapter, a conference paper explaining the HIL simulation of a DC motor using FPGAs was prepared [101]. It was submitted to Turkish National Committee of Automatic Control 2009 held in Istanbul, Turkey and was accepted for oral presentation. Another conference paper explaining the HIL simulation of the described spindle drive was prepared [102]. It was submitted to International Conference on Electrical Machines and Systems 2009 held in Tokyo, Japan and was accepted for oral presentation.

CHAPTER 6

NON-REAL-TIME HARDWARE-IN-THE-LOOP SOLUTION UTILIZING A HYBRID ARCHITECTURE

6.1 Introduction

While the solution presented in Chapter 5 focuses on achieving high performance and connectivity, it lacks the simulating non-linear, complex plants with many states and inputs. To address this issue, another solution utilizing a combination of different hardware platforms and relevant software working in tandem is proposed.

In this solution, a microprocessor bearing board, namely the Atmel NGW100 Network Gateway Kit, takes over the duty of performing the simulation calculations. This is a general purpose development board for Atmel's AVR32 series 32-bit microprocessor CPU AT32AP7000. It has a number of useful storage, communication and I/O devices. Running a Linux distribution specifically tailored for the device, it provides all the drivers necessary to use the peripherals it carries.

Quoting from the microprocessor datasheet [103]; "the AT32AP7000 is a complete System-on-chip application processor with an AVR32 RISC processor achieving 210 DMIPS running at 150MHz. AVR32 is a high-performance 32-bit RISC microprocessor core, designed for cost-sensitive embedded applications, with particular emphasis on low power consumption, high code density and high application performance".

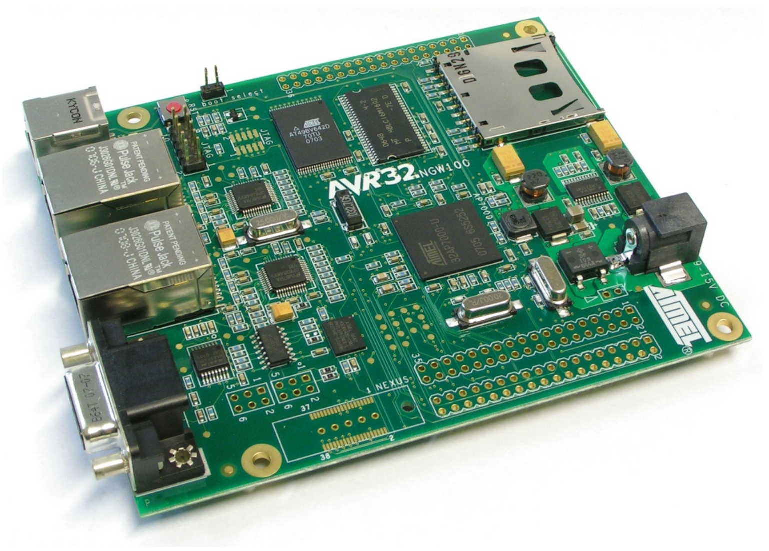


Figure 6.1: Atmel NGW100 Network Gateway Kit [89]

Apart from the hardware features, numerous useful software packages are readily provided on the NGW100 or can be incorporated into the Linux kernel or file system, whichever is necessary and/or applicable. The services available for use on the NGW100 include, but are not limited to; serial port console, FTP and HTTP servers, file sharing and packet routing services, device drivers for the various peripherals on the AT32AP7000 processor and numerous system management utilities provided by the operating system.

The NGW100 is connected to an FPGA board, the DE1, via digital I/O pins. This board carries multiple, configurable instances of the interface emulators explained in Section 5.2.3 and serves as the interface between the simulation and the controller. On the other end, the NGW100 is connected to a personal computer using an Ethernet cable (which serves to transfer the source codes defining the plant behavior as well as state variables during runtime) and an RS-232 cable (which is used for accessing the terminal service for managing the simulation execution and other tasks on NGW100). The software package provided for the PC here is the main tool to be manipulated by the user. One program within the package is used for defining the plant (along with state equations, manipulation inputs, disturbances) and a number of parameters regarding the simulation process. The other program is used to manage the simulation process. The desired set of variables inside the simulation can also be viewed during runtime and saved to the PC hard drive if desired. Figure 6.2 shows the block diagram of this system.

The definition of the plant is done by the user by specifying the state variables, manipulation inputs and disturbance sources. Desired constants and intermediate variables that will aid in defining the equations which govern the system can also be defined. The user also provides the governing equations, as well as any helper functions (for such calculation tasks that are used multiple times in the equations), in the C++ programming language. The use of a programming language instead of mathematical sentences allows for easier inclusion of conditional (decision) statements (e.g. if-then-else and switch/select/case blocks), whenever required, inside the equations.

Once all the information defining the plant is provided, the software package creates necessary C++ declarations for the variables and encapsulates the provided functions; generating a number of source files. These are then added to a previously prepared library containing the necessary functions for performing a simulation. The resulting source collection is used to compile an executable file for the NGW100 that will perform all the tasks necessary for simulation: solution of the state equations, receiving controller inputs, generation of sensor emulator data, reporting and so on. After this executable is sent to the NGW100, the simulation can be run as desired.

Due to the computational capabilities of the NGW100, this solution does not claim real-time performance. Instead, a non-real time operating scheme that treats the controller as the timing master is employed, similar to the scheme explained in

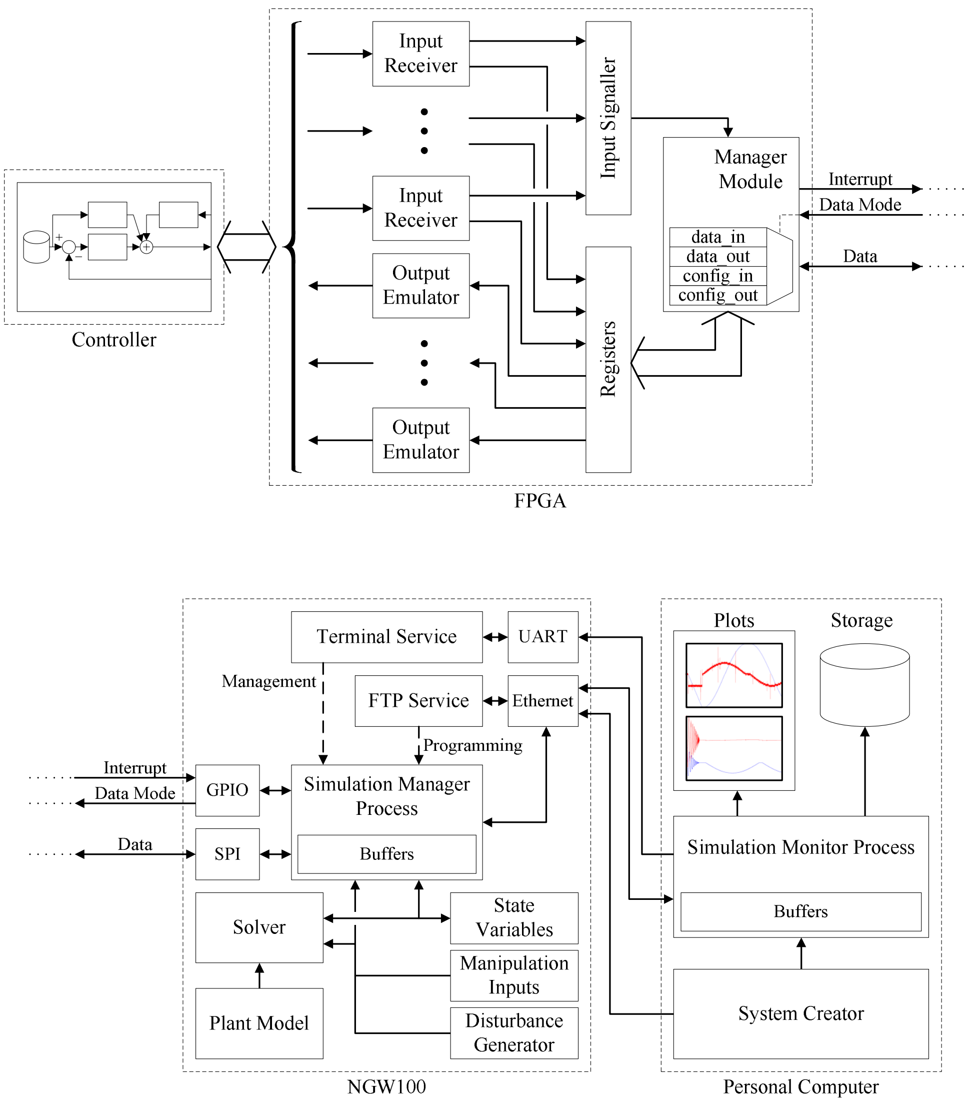


Figure 6.2: Block diagram of the microprocessor-based simulator

Section 4.3.3. The sampling time of the controller is specified ahead of the simulation. The timer source for the controller then needs to be adjusted to allowthe simulator to complete the calculations between two commands. Once these steps are done, the time between two control loops is treated by the controller as if it is equal to the sampling time and all calculations are executed accordingly.

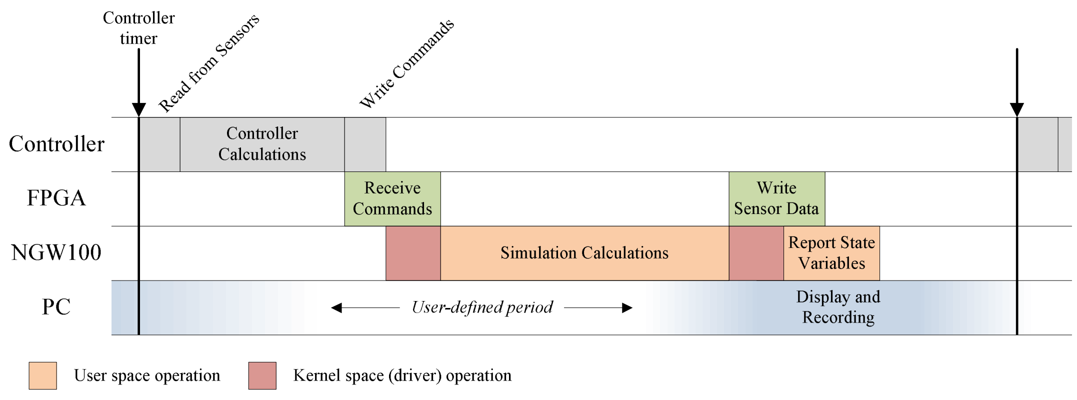


Figure 6.3: Operation timeline for microprocessor-based solution

6.2 Modeling

A 3-axis CNC vertical milling center is selected as the simulated plant for the case study. The dynamics of the three axes, including friction forces, backlash and forces generated due to cutting process are the focus of this simulation application. On the other hand, spindle and thermal dynamics, as well as other systems such as coolant/lubricant pumps and chip removal mechanism are excluded.

The following sections discuss the details of the system model and implementation of this solution.

6.2.1 Plant Model

For the purposes of modeling, a CNC machining center available at the machine shop located in Mechanical Engineering department of Middle East Technical University is taken as reference. It is a First MCV-1100 3-Axis CNC Machining Center by Long Chang Machinery Ltd. Co., equipped with automatic tool changer, coolant and chip removal systems.

The axes of the machine are all mounted on friction (hydrostatic) guideways, and are driven by servomotors via ball screws. The *x*-axis carrying the cart (a.k.a. "table") on which the workpiece is mounted is illustrated in Figure 6.4 and the *y*-axis carries the entire *x*-axis assembly. Housed on the column is the z-axis assembly, carrying the entire headstock (main spindle shaft, motor and tool changing mechanism) (Figure 6.5).

The equation of motion for the x-axis cart can be written as

$$\ddot{x} = \left(m_x + m_w\right)^{-1}\left[F_{s,x} - F_x - F_{f,x}\,\mathrm{sgn}(\dot{x})\right] \tag{6.1}$$

where m_w stands for the mass of the workpiece, m_x is the mass of the cart, F_x is the cutting force on the axis, $F_{f,x}$ is the friction force (dry) and $F_{s,x}$ is the force exerted on the table by the ball screw nut. The equation of motion for the ball screw is

$$\ddot{\theta}_x = \left(J_x\right)^{-1}\left[T_{m,x} - \frac{h_{s,x}}{2\pi\eta_{s,x}}F_{s,x} - \left(b_x\left|\dot{\theta}_x\right|^{\frac{2}{3}} + T_{f,x}\right)\mathrm{sgn}\left(\dot{\theta}_x\right)\right] \tag{6.2}$$

where J_x is the total moment of inertia of the ball screw and rotor, $T_{m,x}$ is the torque applied by the motor, $T_{f,x}$ is the dry friction torque on the ball screw and rotor, $h_{s,x}$ is the screw lead and $\eta_{s,x}$ is the ball screw efficiency. When backlash exists in the ball screw assembly, equations (6.1) and (6.2) are coupled together with the equation

$$F_{s,x} = \begin{cases} k_x\left(d_x - \dfrac{D_x}{2}\right) & ,d_x > \dfrac{D_x}{2} \\ 0 & ,\left|d_x\right| < \dfrac{D_x}{2} \\ k_x\left(d_x + \dfrac{D_x}{2}\right) & ,d_x < -\dfrac{D_x}{2} \end{cases} \tag{6.3}$$

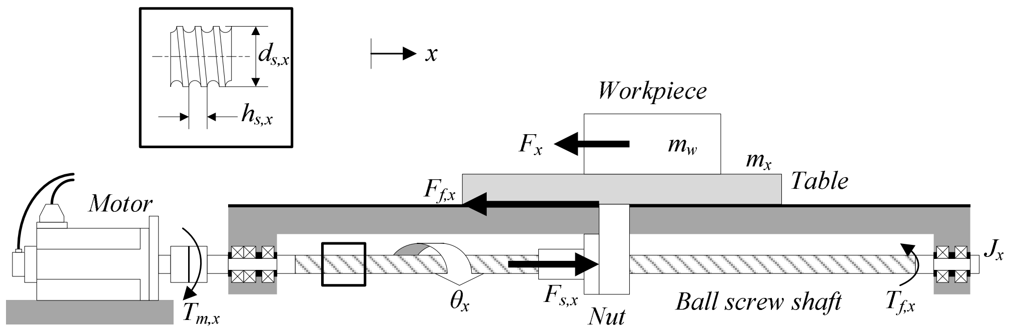


Figure 6.4: X-axis feed drive for CNC machining center

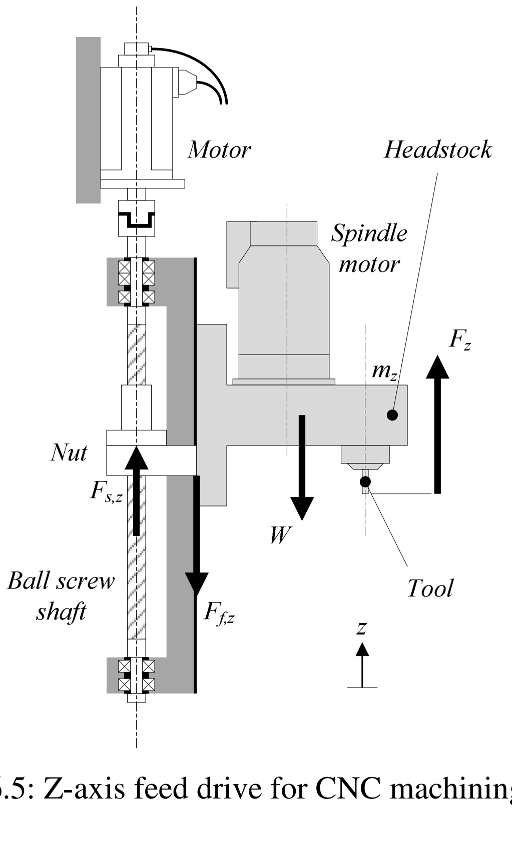


Figure 6.5: Z-axis feed drive for CNC machining center

where

$$d_x = \frac{h}{2\pi}\theta_x - x_x \tag{6.4}$$

as explained in Section 3.2.1.3. If the ball screw is assumed to be backlash-free, however, these equations can be reduced to a single equation of motion that uses an equivalent set of parameters. Using (6.1) and (6.2) one gets

$$\ddot{\theta}_x = \left(J_{eq,x}\right)^{-1}\left[T_{m,x} - \frac{h_{s,x}}{2\pi\eta_{s,x}}F_x - \left(b_x\left|\dot{\theta}_x\right|^{\frac{2}{3}} + T_{f,eq,x}\right)\operatorname{sgn}\left(\dot{\theta}_x\right)\right] \tag{6.5}$$

Here, the equivalent inertia $J_{eq,x}$ is defined as

$$J_{eq,x} = J_x + \frac{h_s^{\,2}}{4\pi^2\eta_s}\left(m_x + m_w\right) \tag{6.6}$$

Figure 6.6: First MCV-1100 3-Axis CNC Machining Center

Since the table's position is linearly dependent on the angular position of the ball screw under no-backlash condition, it immediately follows that the velocities are also linearly dependent and $\operatorname{sgn}(\dot{x}) = \operatorname{sgn}(\dot{\theta})$ holds. Hence, utilizing equations (6.1) and (6.2), the equivalent dry friction $T_{f,eq,x}$ can be simply written as

$$T_{f,eq,x} = \frac{h_{s,x}}{2\pi\eta_{s,x}} F_{f,x} + T_{f,x} \tag{6.7}$$

The equations of motion regarding the *y*- and *z*-axes can be similarly obtained as

$$\ddot{y} = \left(m_y + m_x + m_w\right)^{-1}\left[F_{s,y} - F_y - F_{f,y}\operatorname{sgn}(\dot{y})\right] \tag{6.8}$$

$$\ddot{\theta}_y = \left(J_y\right)^{-1}\left[T_{m,y} - \frac{h_{s,y}}{2\pi\eta_{s,y}} F_{s,y} - \left(b_y\left|\dot{\theta}_y\right|^{\frac{2}{3}} + T_{f,y}\right)\operatorname{sgn}\left(\dot{\theta}_y\right)\right] \tag{6.9}$$

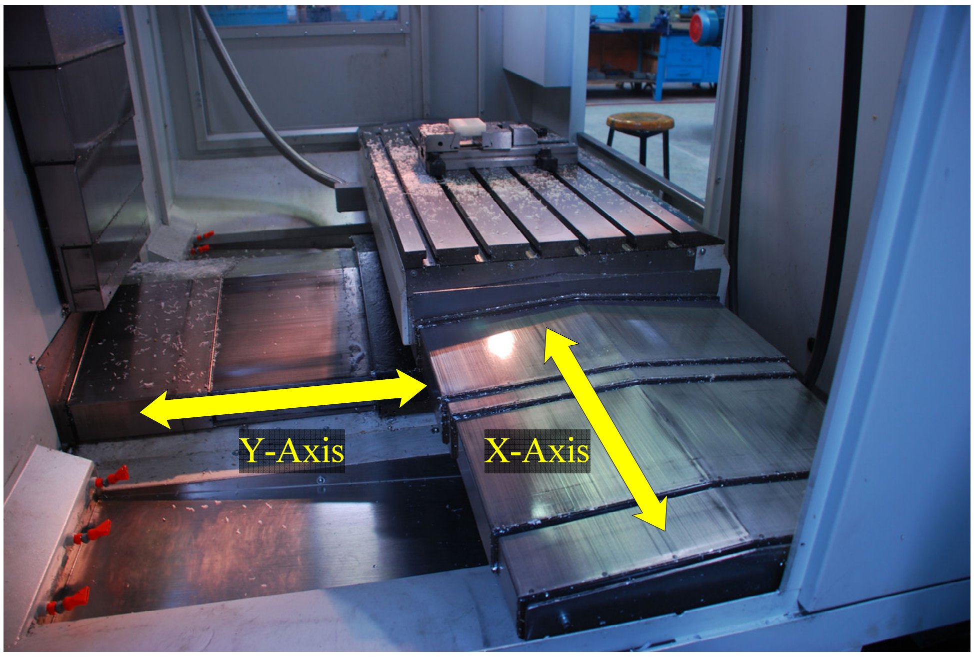


Figure 6.7: Horizontal axes of the CNC machining center

$$\ddot{z} = (m_z)^{-1}\left[F_{s,z} + F_z - F_{f,z}\,\mathrm{sgn}(\dot{z}) - W\right] \tag{6.10}$$

$$\ddot{\theta}_z = (J_z)^{-1}\left[T_{m,z} - \frac{h_{s,z}}{2\pi\eta_{s,z}}F_{s,z} - \left(b_z\left|\dot{\theta}_z\right|^{\frac{2}{3}} + T_{f,z}\right)\mathrm{sgn}\left(\dot{\theta}_z\right)\right] \tag{6.11}$$

Under no-backlash condition, these can be expressed in a simpler form similar to (6.5) as

$$\ddot{\theta}_y = \left(J_{eq,y}\right)^{-1}\left[T_{m,y} - \frac{h_{s,y}}{2\pi\eta_{s,y}}F_y - \left(b_y\left|\dot{\theta}_y\right|^{\frac{2}{3}} + T_{f,eq,y}\right)\mathrm{sgn}\left(\dot{\theta}_y\right)\right] \tag{6.12}$$

$$\ddot{\theta}_z = \left(J_{eq,z}\right)^{-1}\left[T_{m,z} + \frac{h_{s,z}}{2\pi\eta_{s,z}}(F_z - W) - \left(b_z\left|\dot{\theta}_z\right|^{\frac{2}{3}} + T_{f,eq,z}\right)\mathrm{sgn}\left(\dot{\theta}_z\right)\right] \tag{6.13}$$

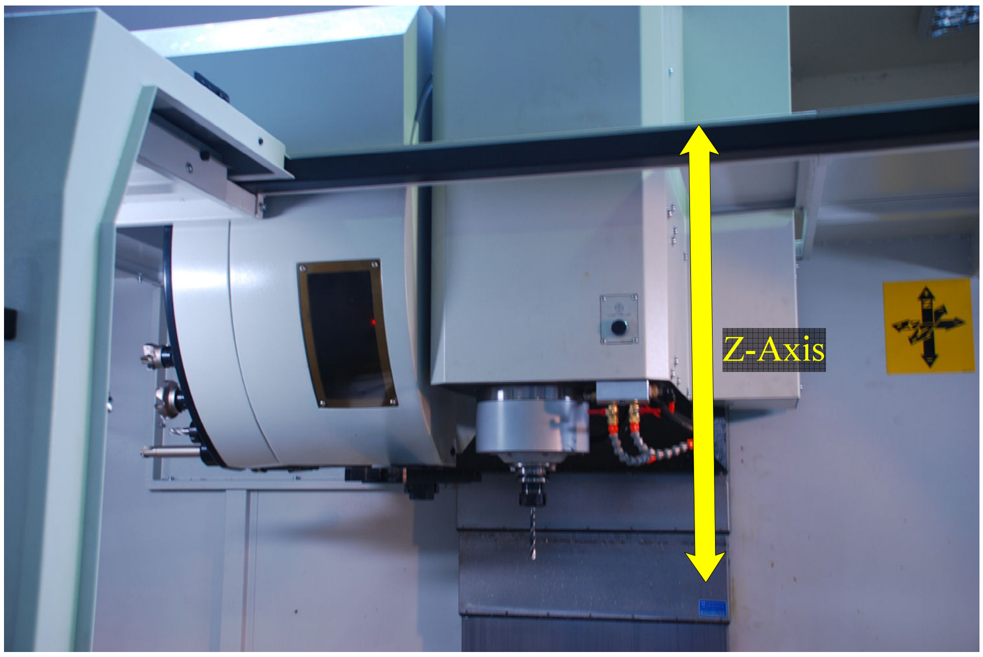


Figure 6.8: Vertical axis of the CNC machining center

Note that in equation (6.13) regarding the *z*-axis drive, the weight of the headstock assembly (*W*) is also included. The feed-drive axes are driven by Fanuc α Series AC Servo Motors, while the spindle motor is a Fanuc α Series AC Spindle (Induction) Motor. As specified in the descriptions manual [104], the speed-torque characteristics of the servo motors have a linearly decreasing tendency in the torque region up to the rated speed. Beyond this point, the motor enters the constant power region, similar to the model explained in Section 3.2.2 (Figure 6.9). The torque envelope of the motor, T_{max}, is then

$$T_{max} = \begin{cases} T_r + m_T\left|\omega\right| & ,\left|\omega\right| < \omega_r \\ \dfrac{P_r}{\left|\omega\right|} & ,\left|\omega\right| \geq \omega_r \end{cases} \tag{6.14}$$

where T_r and ω_r represent the rated torque and rated speed, respectively. T_r' is the torque produced by the motor and P_r is the power output, both at the rated speed, while $m_T = (T_r - T_r') / \omega_r$ and $P_r = T_r'\omega_r$. Torque applied by the motor as response to a torque command T^* is calculated from equation (3.23). Finally, Table 6.1 shows the numerical values of the parameters defining the plant collected from the machine and motor operating manuals.

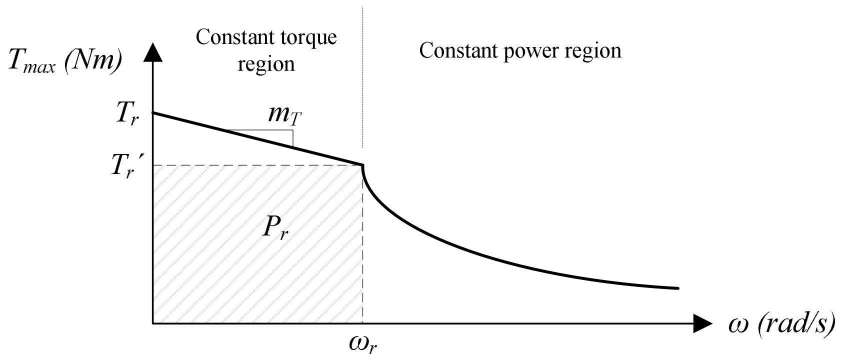


Figure 6.9: Torque capability curve for CNC machining center axis motors

Table 6.1: Plant parameters for CNC machining center

Parameter	**Symbol**	**Unit**	***x***	***y***	***z***
Mass	m	kg	130	331.97	260
Dry friction force	F_f	N	200	200	200
Moment of inertia	J	kg m^2	7.9941×10^{-3}	16.4838×10^{-3}	19.7446×10^{-3}
Dry friction torque	T_f	N	1.1	1.5	2.1
Viscous friction coefficient	b	Nms/rad	0.0005	0.0005	0.0005
Equivalent moment of inertia	J_{eq}	kg m^2	0.00834	0.01737	0.02044
Equivalent dry friction	$T_{f,eq}$	N m	1.435	1.835	2.435
Ball screw lead	h_s	m	0.010	0.010	0.010
Ball screw efficiency	η_s	-	0.95	0.95	0.95
Rated torque	T_r	N m	12	22	30
Rated speed	ω_r	rad/s	209.44	209.44	209.44
Rated power	P_r	W	2,094.4	3,769.9	4,398.2
Torque-speed slope	m_T	Nms/rad	-0.00955	-0.01910	-0.04297
Encoder resolution	-	pulses/rev	10,000	10,000	10,000

6.2.2 Disturbance Model

The disturbance in a machining center can be attributed mainly to the cutting forces generated during the machining process. These forces originate from the feed motion on the axes, chip removal by the tool cutter edges and eccentricity of the tool axis. In the ideal case, these forces must be derived based on a simulation of the actual cutting process. However, such a simulation would involve not only the dynamics of the chip removal process, but also a complete solid model of the workpiece in order to track the removed and remaining material. In addition to the tremendous computational resources needed for such a simulation, modeling of the removal process is out of the scope of this thesis and is not included in the studies.

Instead of the actual disturbance model, an approximate model that generates a typical disturbance force for the cutting process is used. Based on the provided information, form can be composed using a mean value and a higher harmonic component of the spindle rotation, reflecting the feed and chip removal, and the first harmonic component of the spindle rotation, reflecting the eccentricity. The actual frequency of the higher harmonic component depends on the number of cutting edges on the tool, N_c. During each revolution of the spindle, every edge engages and disengages the workpiece once; generating force with frequency N_c times that of the spindle revolution. The disturbance force as a function of simulation time can then be written as follows:

$$F_d^*(t) = F_{d,\max}\left(c_1 + c_2 \frac{\sin(N_f \omega t) + 1}{2} + c_3 \frac{\sin(\omega t + \phi) + 1}{2} \right) \qquad (6.15)$$

where $F_{d,max}$ is the maximum disturbance, ω is the spindle speed and ϕ is the angular difference between the first tool edge and spindle eccentricity. Coefficients c_1, c_2 and c_3 are selected to adjust the weight of the mean and two harmonic components and satisfy the following conditions:

$$c_1 + c_2 + c_3 = 1 \qquad (6.16)$$

$$0 \le c_1 \le 1, \quad 0 \le c_2 \le 1, \quad 0 \le c_3 \le 1 \qquad (6.17)$$

After the form of the disturbance force is known, the direction in which it is applied needs to be determined. Furthermore, in an actual process, the disturbance is only observed during chip removal. However, this is not included in the simulation. Instead, the reference trajectory for the simulated process is used to determine the direction and existence of disturbance. For any axis, the disturbance F_d is

$$F_d = \begin{cases} -F_d^* \operatorname{sgn}(v) & , \operatorname{sgn}(v)\operatorname{sgn}(v^*) = 1 \\ 0 & , \operatorname{sgn}(v)\operatorname{sgn}(v^*) \neq 1 \end{cases} \qquad (6.18)$$

where v is the plant velocity and v^* is the reference velocity. The above equation ensures that the disturbance is always in the opposite direction of the feed movement and feed is in the direction of the uncut material. When the directions of the plant and reference command do not match, the disturbance force becomes zero as no cutting is expected to occur in this case. This model also prevents the disturbance from banging between extremes in two directions in cases where the velocity fluctuates around zero due to controller effort or numerical errors.

For the application, three sets of disturbance forces are used for different scenarios: light, medium and heavy machining. The forces in each axis are selected such that for these machining types, the maximum resultant force is 1kN, 2.5kN and 4kN, respectively. Figure 6.10 shows an example disturbance form for a single axis, generated for 1kN resultant force.

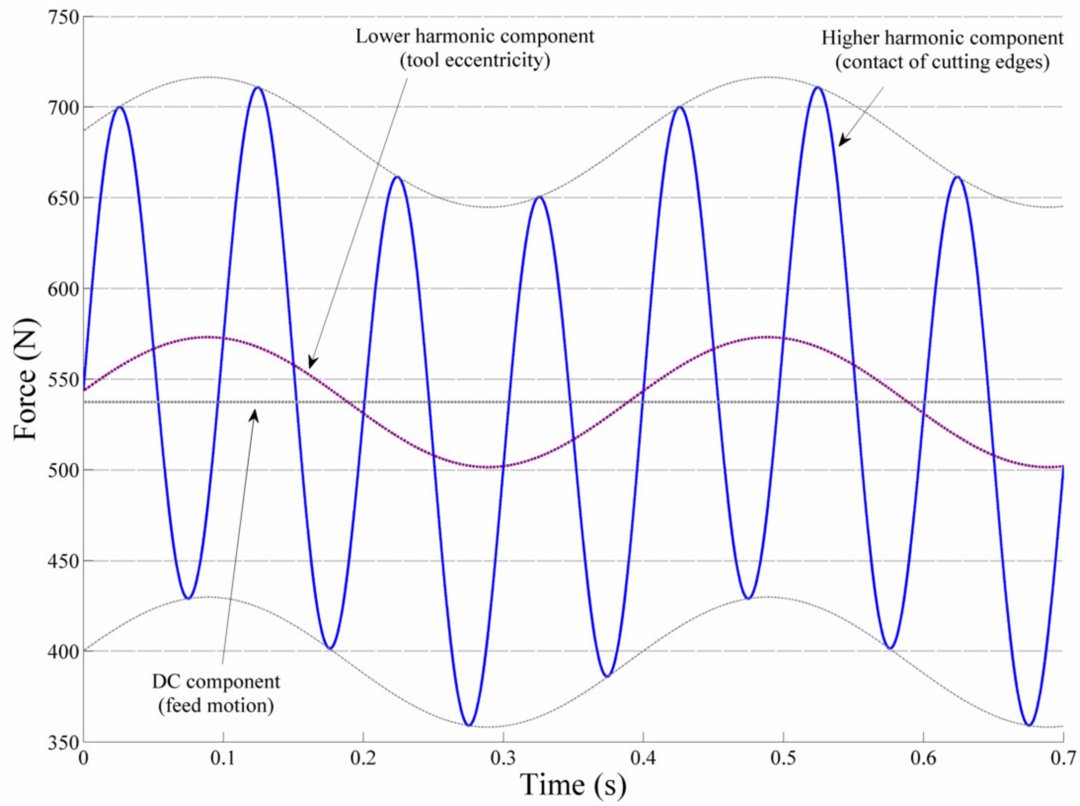


Figure 6.10: Example of light cutting force disturbance on single axis

6.3 Implementation

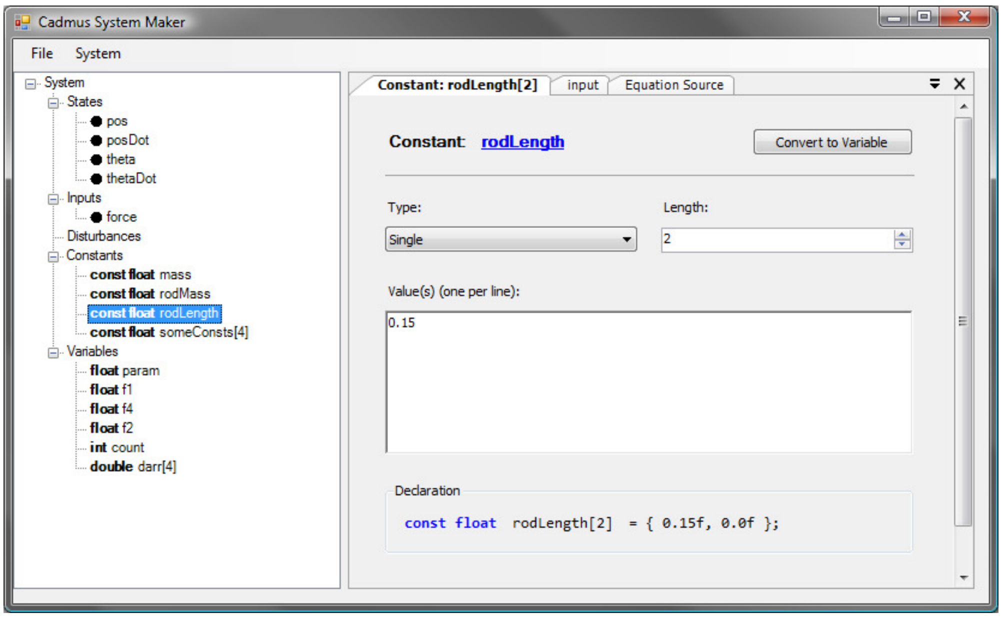
The implementation of the proposed solution involves the development of necessary codes for the PC, NGW100 and FPGA platforms. In addition to their defined duties, these codes are also required to perform necessary communication and remain synchronized for successfully performing the simulation. The following sections explain the details of these codes, discussing the primary features required from each platform and methods for meeting these requirements.

6.3.1 PC Software

The PC software package aims to provide the user with the tools they will directly interact with in order to design and execute an HILS. Two programs are contained in this package: System Maker and System Monitor.

The System Maker program allows the user to design the plant to be simulated. The program's GUI presents the user with a tree view displaying the states, inputs, disturbances, constants and other intermediate variables of the plant; as well as its state equations and other settings related to the simulation process (Figure 6.11). A tabbed document interface provides the editing area for the selected properties.

The user may start the design from scratch or open an existing System Maker file for modification. In any case, as many states as desired can be added, existing states may be removed or renamed from the interface. Inputs, disturbances, and other parameters can be similarly edited. The user is able to select the sources of the inputs, i.e. what type of command receiver (digital, PWM, etc.) will be employed to obtain a certain input. Likewise, the disturbance sources can also be specified as mathematical functions (which can be evaluated on the NGW100 during run-time or on the PC and then transferred via Ethernet), pre-generated patterns (saved in binary files and read from NGW100 storage) or pseudo-random



Constant: rodLength

Type: Single

Length: 2

Value(s) (one per line):

```
0.15
```

Declaration

```
const float rodLength[2] = { 0.15f, 0.0f };
```

Figure 6.11: Screenshot from the System Maker tool

generators with specified mean, minimum, maximum and distribution properties. States, inputs and disturbances are fixed to non-array, single-precision floating point data types. On the other hand, constants and intermediate variables can be declared as single- or double-precision floating point values, bytes or short, normal and long integers. They can also be declared as arrays of any length. Additionally, all the parameters except inputs and disturbances can be assigned with the desired initial values.

The definition of the plant's state equations is made by expressing them using the C++ programming language. The user is presented with a formatted text box and simply needs to type in the equations for the derivatives of each plant state. These equations are encapsulated inside function that the NGW100 can use. Although the function header is pre-defined, the user is free to use the names they specified for states or other parameters as aliases.

Aside from the state equation, it is possible to specify pre-solve and post-solve functions. Unlike the state equation, these functions are not passed to the solver routine, which may call the function more than once due to its algorithm. These functions allow the user to specify linear relations between states, perform logical operations, store past state values and such tasks that need not or should not be passed to the solver. An even greater flexibility in plant definitions is thus provided.

After the plant design is complete, System Maker generates the necessary source files from the provided information. These are combined with other source files for the NGW100 platform for cross-compilation, which is explained in the following chapter. Once compiled, the binary files are transferred to the NGW100, ready to perform the simulation.

The System Monitor is a tool for displaying the state variables in the simulation as well as initiating and stopping the process. During the simulation, the NGW100 reports the state, input and disturbance values are received via Ethernet. The Monitor holds the value history in the computer memory, plotting their change over time using on-screen graphs (Figure 6.12). If desired by the user, these values can be stored in a file using a format suitable for importing into other software packages. The Monitor is further capable of sending the necessary commands to start and stop the simulator to the NGW100 over serial port and Ethernet.

The plant definition file generated using the System Maker is also used by the System Monitor to determine the variables in the plant and whether each one of them is reported by the NGW100 or not. Based on this information, the user is presented with a display setting window, using which the placement of individual plots on the screen, as well as trace colors can be adjusted. With these settings, the user can adjust the screen layout to their convenience.

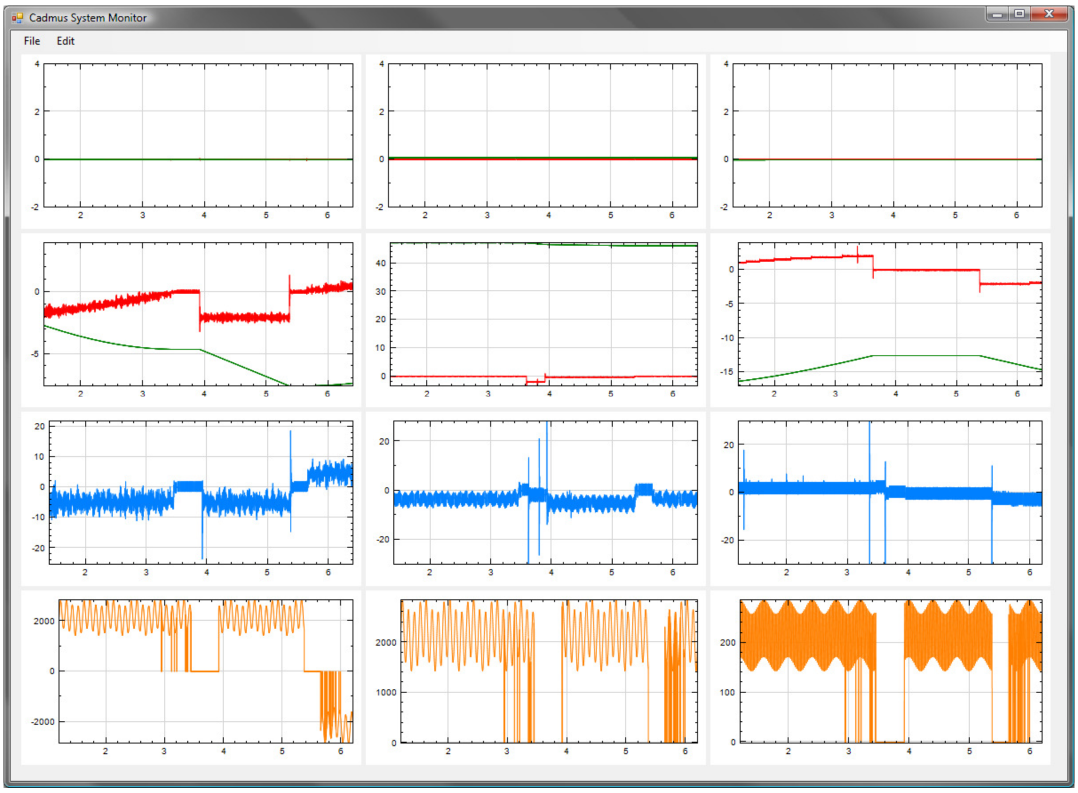


Figure 6.12: Screenshot from the System Monitor tool

The entire PC software package is developed using the C# programming language, dependent on the Microsoft .NET Framework. It can run on Windows XP, Vista and 7 operating systems. The System Monitor also uses the NPlot Charting Library [105] to draw the graphs.

6.3.2 NGW100 Software

Since the NGW100 is the main workhorse of the simulator, it is the most essential one of the hardware platforms used. In addition to the simulation calculations themselves, its tasks include transfer of data to and from the FPGA in order to receive the inputs coming from the controller and emulate the sensor outputs, as well as reporting of the state variables that belong to the simulated plant. The software developed for NGW100 in order to accomplish said tasks can be separated into two main parts: simulator program and FPGA interface driver module.

After the simulated plant is defined by the user and necessary sources and/or executables are placed on the NGW100, the simulator program is started via commands received from the RS-232 port. The start-up process involves the opening of the FPGA interface driver, initialization SPI module on the board, setting of the initial values of the state variables, calculation of derived constants and finally establishing socket connection to the user's PC over Ethernet cable.

The solution process is triggered by an interrupt signal originating from the write operation of the tested controller. Passing through the FPGA interface, this signal tells the NGW100 to acquire the inputs from the FPGA on which they are registered. Using the I/O pins on the NGW100 board, the FPGA is put into "read" mode and the inputs are received via high-speed (10Mbit/s) SPI protocol. The disturbance inputs to the plant, if any, are also generated, read from storage or obtained from buffers that are filled by the user's PC. All the inputs and plant states are then used to solve for the state variables for the next time step. An implementation of the 4th order Runge-Kutta solver method is then employed to solve for the state variables for the next time step, using the inputs and current states. After the calculation, commands for the sensor emulators on the FPGA board are generated and sent, this time putting it in "write" mode and again sending the data over SPI. Finally, the inputs and state variables are written into a buffer to be sent to the PC for display and recording. The NGW100 then returns to a waiting state, ready to receive the next interrupt signal from the FPGA.

To increase the responsiveness and performance of the simulator, multiple execution threads are used. The main process responsible for initialization of the program components is the first thread. Handling the commands that can be received over the RS-232 is also done here. The second thread performs the solution of the state equations and the read/write operations to the FPGA board, as well as filling the buffer containing the state values to be sent to the PC. Its operation is triggered by the driver module. The last thread takes care of the communications with the PC. Its operation is interrupted and blocked by the simulator thread whenever necessary, only to be resumed when CPU time is available again, ensuring the simulation has the highest priority.

In order to accomplish its tasks, the simulator program also utilizes the FPGA interface driver; a loadable Linux kernel module which provides low-level access to the system resources without the need to recompile the kernel [106]. The NGW100 is set up to automatically load this module on startup, which calls the necessary routines that enable the use of the GPIO chip. Since CPU interrupts are only available to code executing in the kernel space in Linux, it also registers itself to receive interrupts coming from GPIO pins. Thus, when the simulator program is launched, it can register itself with the driver to be notified of these interrupts, allowing the simulation loop to be initiated when they occur. The module also drives the necessary I/O pins for putting the FPGA board into "read" or "write" states.

The programming tasks for NGW100 are done using the AVR32 Studio IDE [107]. The tools contained in the IDE are capable of interfacing with the board using relevant interfaces for debugging purposes. The main programming language used is C/C++, providing all the facilities of the standard libraries developed for the language as well as those for the Linux operating system. The libraries, executables and other resources developed by the user are cross-compiled for the NGW100 platform and can be easily downloaded via FTP or transferred by means of an SD-Card. Furthermore, since AVR32 Studio is built on top of the Eclipse IDE [108], any plug-ins written for Eclipse are compatible, providing a more user friendly environment to the developer. The code developed on the IDE is cross-compiled for the NGW100 using the tools contained in AVR32 Buildroot [109-110] (a set of compilation tools and utilities for embedded Linux systems).

The source code of the simulator program is prepared in an object-oriented fashion and it is functionally divided into a multitude of classes for better organization and reusability. In addition to these classes, the libraries *stdc++* (C++ standard libraries), *pthread* (POSIX threads library) and *rt* (real-time library) are also employed. The interface driver is written in C language and only uses the Linux kernel libraries.

6.3.3 FPGA Design

The FPGA board serves as the interface between the entire simulation and the controller being tested. Therefore its duties are basically the notification of the simulation of the received commands and emulation of the sensors existing in the plant. The realized digital circuit consists of a communicator/manager module, input reception signaler, data and configuration registers and finally, the receivers and emulators themselves.

The manager module serves as the center of operations on the FPGA. When manipulation commands are received from the controller, they are collected from the relevant registers and the NGW100 is notified by sending an interrupt signal. The commands are then sent over SPI upon the reception of the read instruction from the NGW100. After the simulation calculations are completed, the commands necessary to generate signals at the sensor emulators are received via SPI. These are written into the appropriate registers and the operation loop is thus completed.

When the controller sends manipulation commands, each receiver block raises a signal indicating it has received data. Signals from all the blocks are monitored by the input reception signaler module. When a signal is received, a corresponding flag is raised in an internal register. When all the flags belonging to the receiver blocks used in the simulation are raised, i.e. each block used has received a command; the signaler triggers the manager module and resets the flags. This way, reception of commands on every available receiver block is marked as the initiation of simulation calculations, realizing the “timing master is the controller” scheme.

All the receiver and emulator blocks are connected to two register blocks: data and configuration. While data obviously serve for the storage of input and output data, configuration registers hold information necessary for proper and compatible operation of these blocks if necessary. Such information may include, but is not limited to, PWM periods, encoder pulse per revolution counts, sampling time and so on.

The FPGA would ideally contain multiple instances of various receiver and sensor emulator blocks. The configuration registers can then be used to enable the blocks that are required for the execution of the desired plant simulation. The implementation here, however, only contains the ones necessary to simulate the CNC machining center and does not go into a completely flexible design.

6.4 Results

The functionality of the developed implementation is first performed by connecting constant command generators to the receivers inside the FPGA board. Using digital oscilloscopes and timers available on the platforms, various measurements regarding calculation and communication times were made.

Once correct operation of all hardware platforms is observed, the simulator is connected to a controller also developed on an FPGA platform. The controller is the result of a root locus design procedure and it is loaded with command references for the machining operation for one part an injection mold of a bottle. The simulation of this operation is repeated for different disturbance force magnitudes, as well as under backlash and no-backlash conditions. The results are successfully used both in debugging of the simulator software and optimization of the said controller. Figures 6.14 through 6.21 present samples from the command references and error data gathered during the test processes.

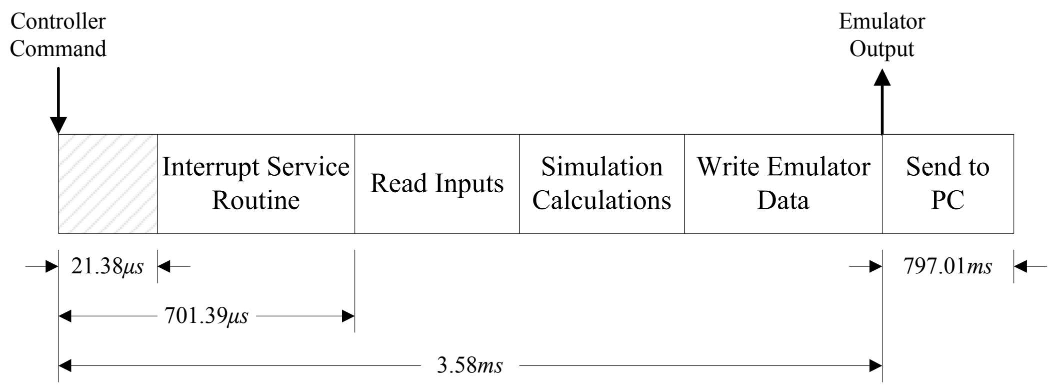


Figure 6.13: Mean times for sub-processes within the NGW100

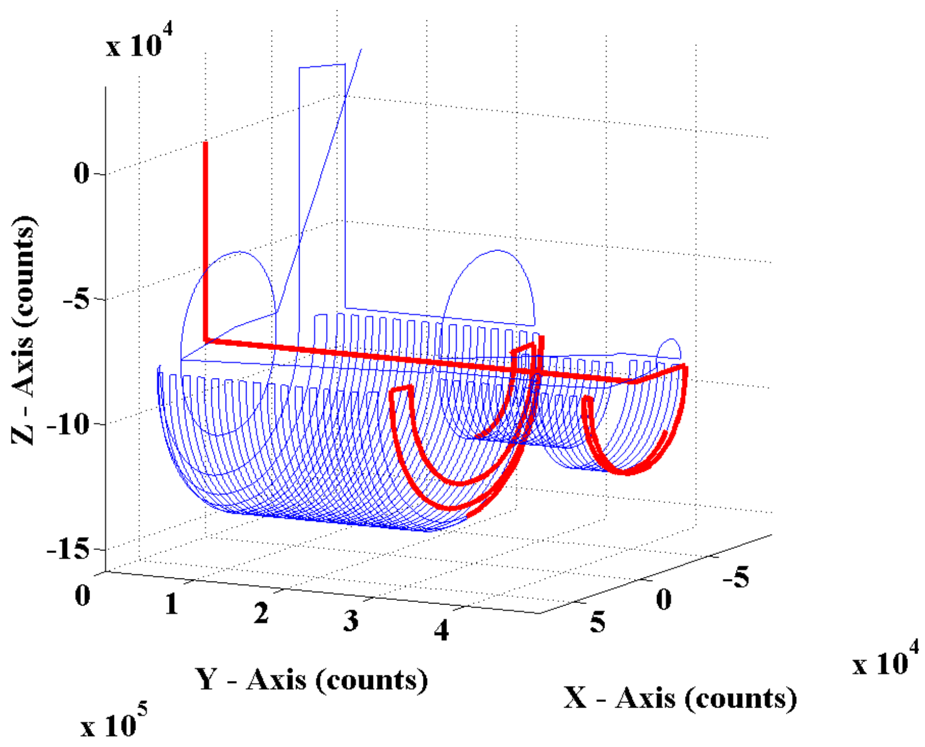


Figure 6.14: Cutting tool trajectory for bottle injection mold

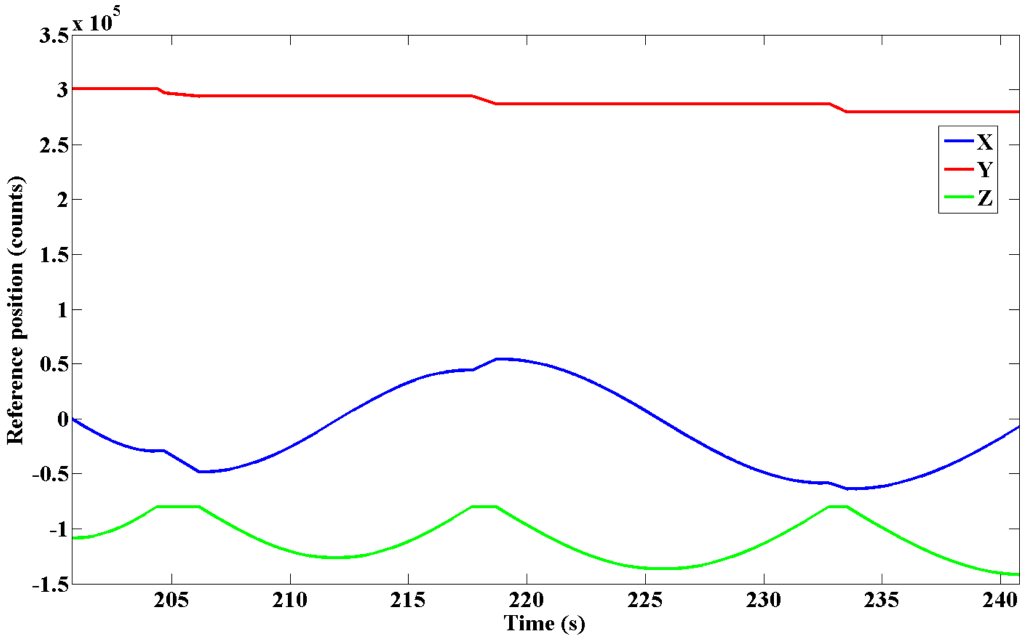


Figure 6.15: Section of axis references for bottle injection mold

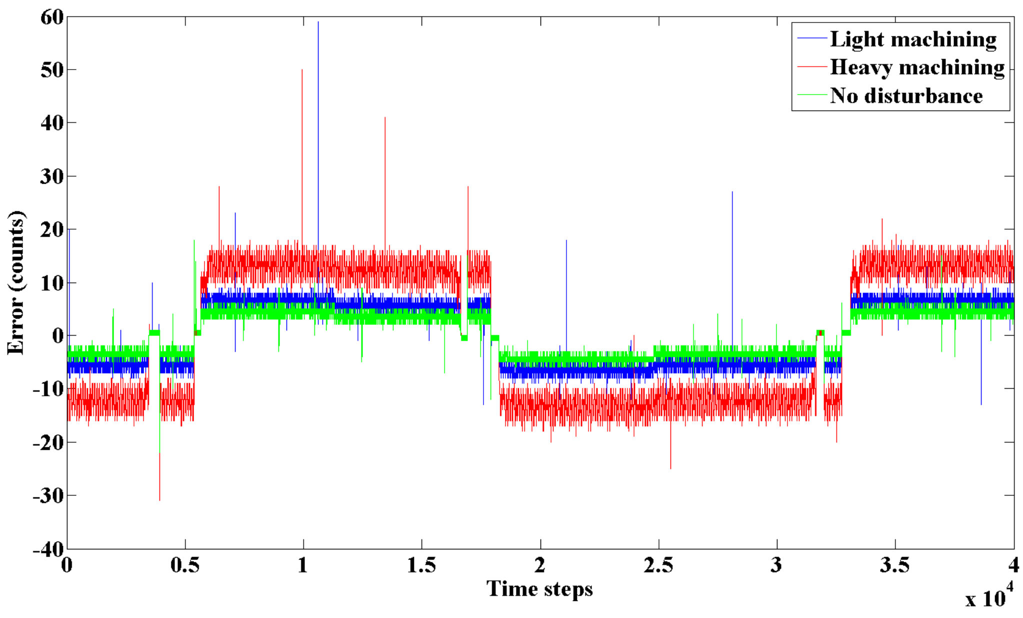


Figure 6.16: Section of *x*-axis motor position error for bottle injection mold

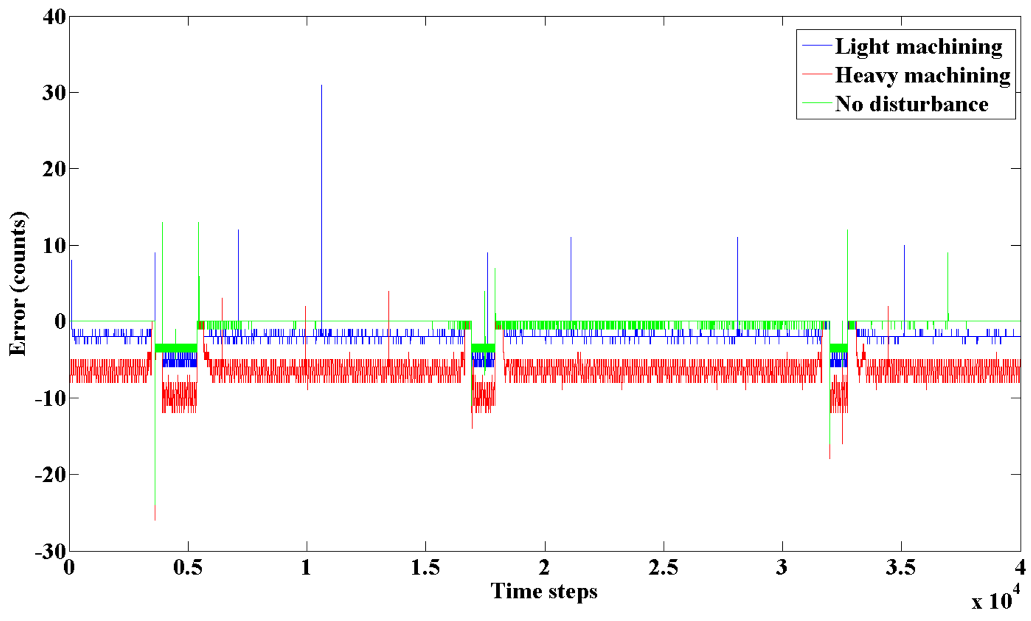


Figure 6.17: Section of *y*-axis motor position error for bottle injection mold

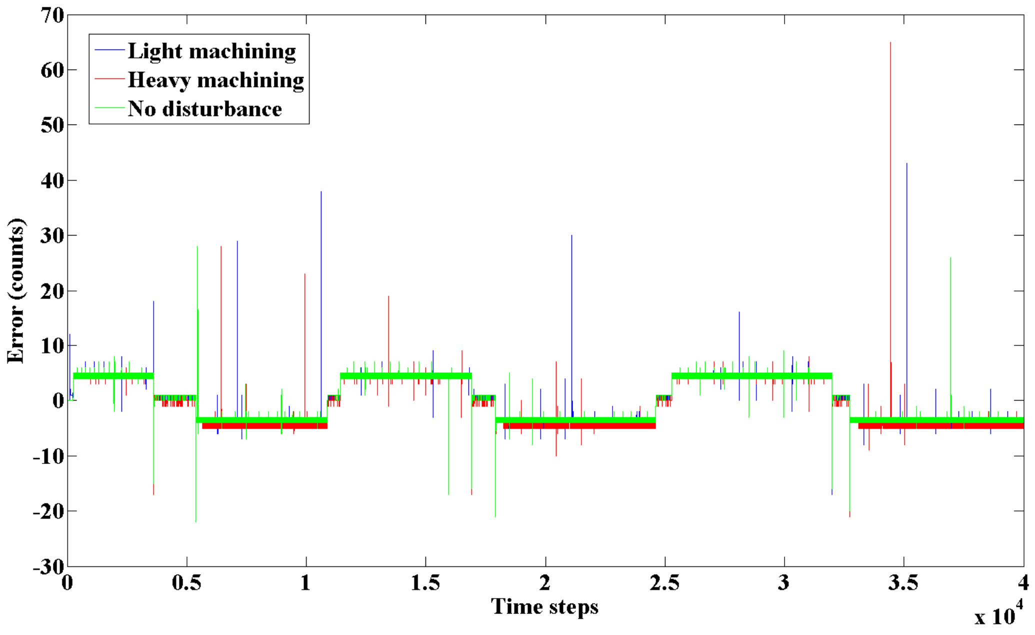


Figure 6.18: Section of *z*-axis motor position error for bottle injection mold

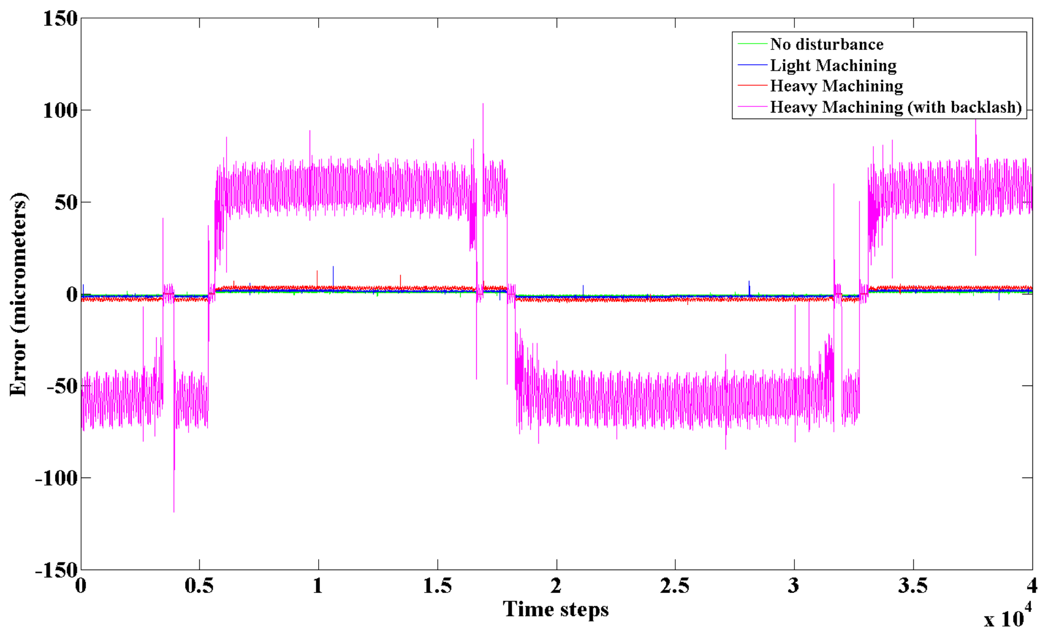


Figure 6.19: Section of *x*-axis cart position error due to backlash for bottle injection mold

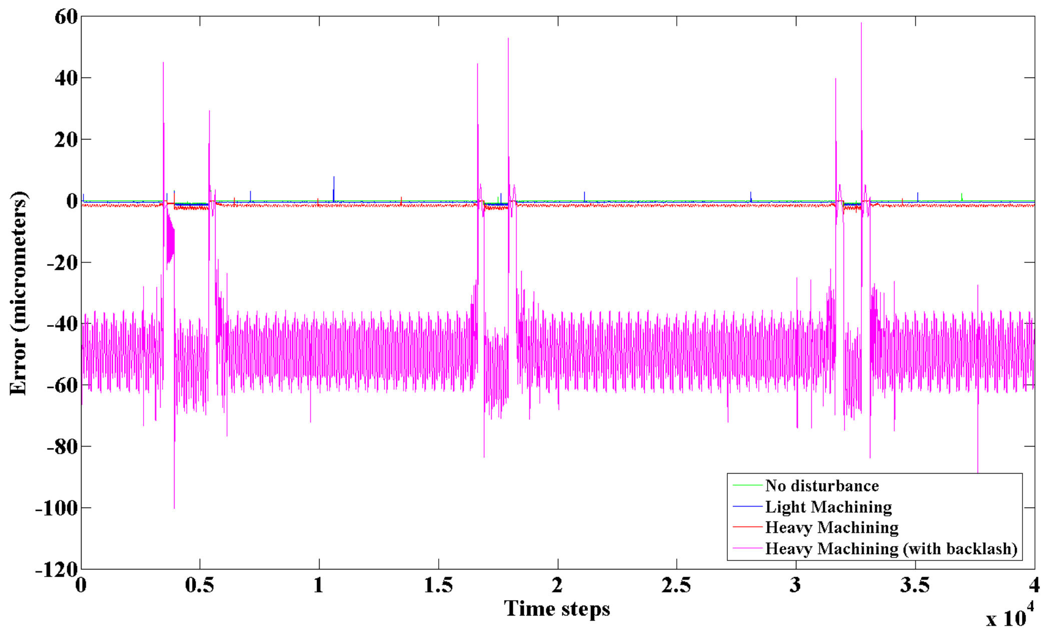


Figure 6.20: Section of *x*-axis cart position error due to backlash for bottle injection mold

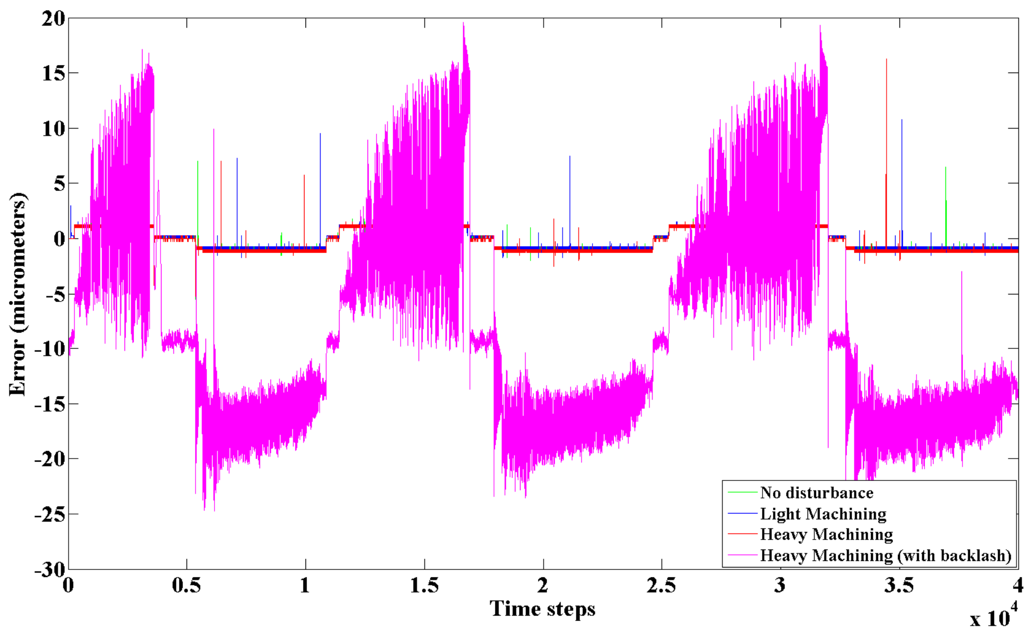


Figure 6.21: Section of *x*-axis cart position error due to backlash for bottle injection mold

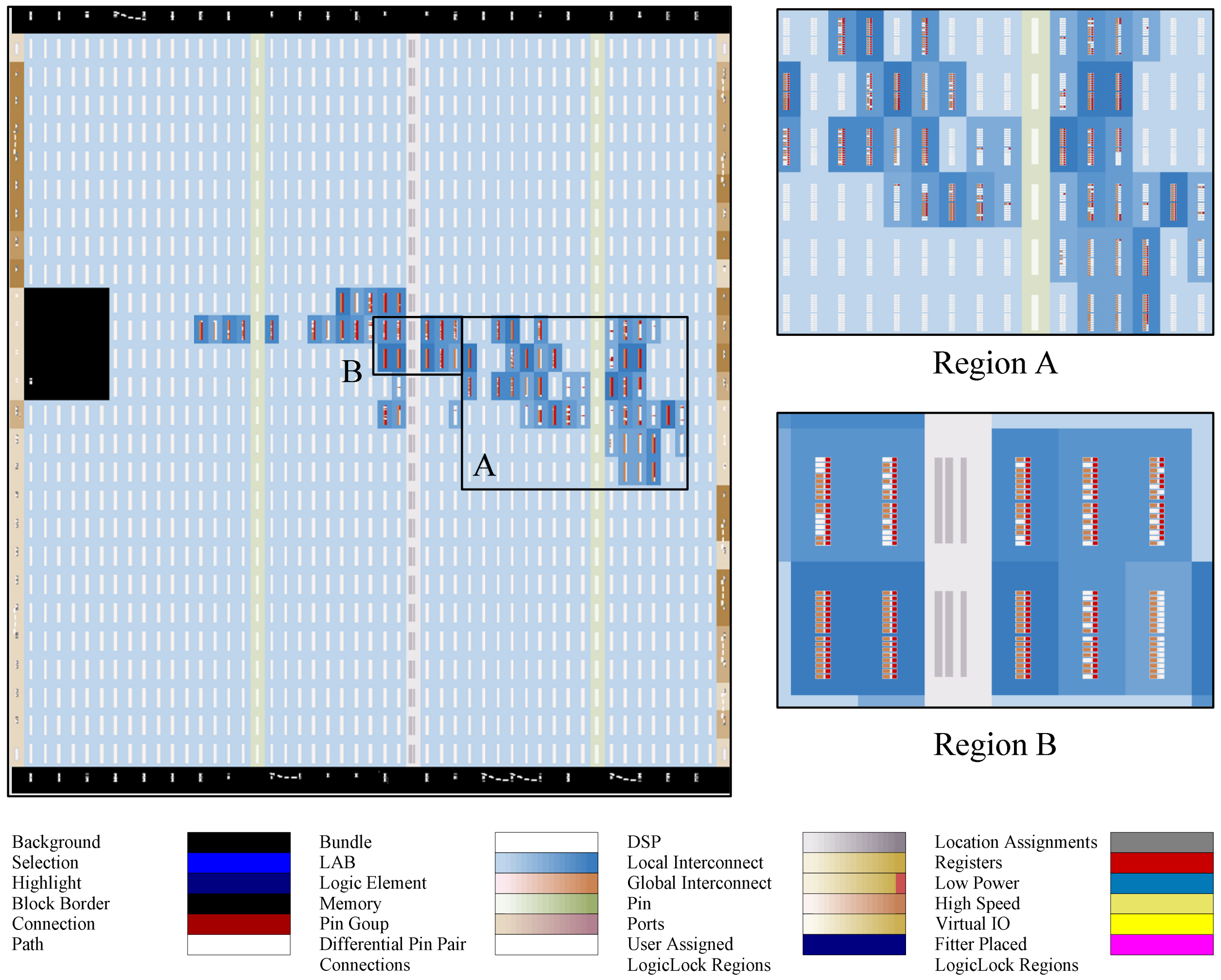


Figure 6.22: Hybrid solution FPGA utilization floor plan

Finally, the resource usage of the software developed for each piece of hardware is investigated. The System Maker and System Monitor applications take up 83kB and 207kB in binaries. When launched, these report 20.5MB and 11.8MB of total memory usage. Saved plant designs take up no more than 5kB, although the size of saved plant designs exact size is dependent on the plant itself. The size of the recorded states depends on the number of states and inputs to the plant, as well as the simulation duration. For the CNC machining center, the recording for a 15-minute cutting operation (including all states, torque commands and disturbances for a total of 18 variables) occupies up to 62MB of storage when saved in binary form or up to ~150MB in text form. These are obviously insignificant memory and storage capacities for today's computers. With all the statically linked libraries, the simulator program and interface driver for the NGW100 are 32.5kB and 5.8kB in size, respectively. The total memory allocated by the simulation processed is limited by 2MB. Taking into consideration the specs of the NGW100 and that an SD-Card with 256MB of storage capacity is used, the platform easily handles the application. Finally, the resource usage on the FPGA is presented in Table 6.2 and Figure 6.22 shows the chip utilization floor plan.

Table 6.2: Hybrid solution FPGA resource usage

Total logic elements	924 (5%)
Total combinatorial functions	591 (3%)
Dedicated logic registers	704 (4%)

6.5 Closure

The CNC machining center application presented here constitutes a full-feature HILS system, complete with controller interfacing, simulation and data recording. Performing all the desired tasks and providing a test bed for development of an actual controller, it demonstrates successful implementation of the techniques discussed in this thesis.

Timings of the various tasks within the implementation show that simulation speeds up to 300Hz are possible, which is unsuitable for real-time applications. Scaled-time operation with the controller as timing master, however, is possible. Upon inspection, it is observed that an important proportion of the calculation durations is spent between the reception of the interrupt signal from the FPGA and initiation of calculations on the NGW100. This delay is caused by the interrupt handling and kernel-to-user space signaling mechanism inside the Linux operating system. Unfortunately, this mechanism cannot be modified by the simulation. In order to minimize this delay, a real-time Linux kernel or another real-time operating system needs to be employed on the simulator platform. At the extreme, it is possible to completely get rid of an "operating system" and run stand-alone simulation calculations. However, the task of including appropriate hardware and communication drivers in the code for such a complicated hardware platform is extremely difficult.

The high deviations in simulation calculation durations make it difficult to emulate sensor signals that are dependent on time, such as the encoder generator. Owing to the specific controller used, which is capable of handling ill-delayed encoder signals, the emulators are configured with lower sampling times in this specific application. This enables the encoder signals to be sent without losing pulses. However, periods during which the pulse generator stops working are also introduced by this modification. The timing of the pulses is also no longer proper. These side effects may cause problems with other controllers, where the measurement of the signal timings is meaningful. While it is true that the use of a real-time operating system will improve the calculation durations, methods for minimizing their effect on signal emulation need to be investigated, perhaps leading to a revised emulator implementation.

Despite the use of a separate processor platform, it is always possible to employ an embedded microprocessor design (such as Nios II) on the FPGA board and eliminate the actual processor. In such a substitution, improved coupling of the simulation with the interface emulators becomes possible. This enables the removal of some additional management modules and relevant code, eliminating a significant portion of the computational delays as well as resulting in a more compact solution. However, there are certain downsides to this change. First of all, even the most basic microprocessor design takes up an important portion of the available resources on the FPGA board (e.g. Nios II uses up to %30 of the logic elements on the DE1 board). Addition of communication interfaces (Ethernet, USB etc.) and an FPU to this design further increase the usage, leaving even less room for the interface emulators. If one further attempts to implement multiple FPUs (commonly utilized via DSP- and SIMD-specific instructions on microprocessors), the resources on the FPGA are rapidly consumed and the advantages of using an embedded processor design are lost. The use of a stand-alone microprocessor platform is therefore more feasible. On the other hand, design of a board which accommodates the microprocessor and the FPGA chip together, as well as all other external resources, is a possibility.

CHAPTER 7

CONCLUSION AND FUTURE WORK

7.1 Conclusion

This thesis is aimed at developing an integrated, self-contained and user-friendly solution to perform HIL simulations. This solution is desired to contain the necessary tools for designing and implementing simulation processes, interfacing the controller hardware at hand with the simulator and also the viewing and recording of the data generated during the simulation for analysis. For this purpose, it proposes the use of a number of hardware platforms that specially designed to perform the simulations and interface with controllers. It also presents numerous software packages that allow the user to define the plant to be simulated, generate the necessary machine codes to realize that simulation and finally execute it.

The first work done is the development of mathematical models of common elements found in dynamical systems. After this, methods for solving these mathematical models (and plants composed of these) numerically in a simulation application are presented.

With the required techniques at hand, an initial attempt at creating an integrated HILS solution is made with the aim of providing an educational tool is made. This tool allows control engineering students to conduct extensive laboratory-like work with the aid of a simple development board and a software package. This facilitates the reinforcement of the material covered in the courses by introducing visual elements and making it easier to understand the controller and plant within cause-effect relationship. The Cadmus package developed during two applications of this concept clearly shows via student feedback that it successfully fulfills its purpose.

With certain elements of an integrated HILS solution provided by the educational solution, the focus is shifted to the performance and connectivity aspects of the simulation. An FPGA chip with its host development board is employed to implement a high-speed solver for simulations. It was demonstrated that although such an approach results in a very fast simulator, it suffers from a trade-off between speed and flexibility. As the simulation of more complex system models involving various mathematical functions and conditional statements are desired, the proposed solver becomes incapable of handling these equations and the need for utilization of a processor unit emerges. On the other hand, the developed system contains sensor emulators and command receivers that enable the interfacing of the simulator with any external controller hardware. This successfully fulfills the connectivity required from such a simulator system.

As the final stage of the studies, a complete HILS system utilizing the successful elements of the previous works is developed. Employing a combination of different hardware platforms with appropriate communication interfaces, this system exhibits all the desired features: flexible controller interfacing, the capability of solving almost any given set of plant equations, display and recording of system states during run-time and all the software tools that the user will need during design and execution of the simulation. Although the resulting system lacks real-time simulation performance, it well demonstrates the working of the proposed setup. With the necessary optimizations and modifications to both hardware and software, it is evident that building a high performance simulator is possible.

One might suggest the use of a personal computer, supported by an I/O card for controller interfacing, in order to perform HIL simulations. Via the use of a real-time operating system, such a platform would have a high computational power owing to modern CPUs (and as of the latest trend, computational use of GPUs). As a matter of fact, such setups are already being used in various HILS applications. However, this approach contradicts with certain goals of the studies of this thesis. First of all installation of a real-time OS on the user's PC is quite intrusive. During the execution of the simulation, all other work on the computer must be ceased; otherwise the user's activity will negatively impact the computational performance. This has a negative impact on the user-friendliness aspect of the solution. Furthermore, modern PC designs utilize many advanced hardware components and interfaces. Typically hidden below several abstraction layers in user applications, high-performance using these components in a real-time application requires careful low-level code design. Also, in contrast to high-performance yet simple board designs, there may be many delay sources and other bottlenecks that cannot be bypassed by any means. Despite the high computational capabilities of the platform, these effects may prevent the realization of a real-time general-purpose simulator. Therefore, a smaller-scale platform similar to the NGW100 remains as a feasible and much simpler solution to the problem.

7.2 Future work

The results obtained from the different solutions proposed in this thesis point out to a number of future work opportunities, some specific to themselves and some that can be generalized to the whole concept.

The results of the educational package show that in order to maximize the efficiency of the application, extensive testing and debugging of such an application is necessary. It is also desirable to have a design tool to be provided to the course instructor or assistants, who may wish to modify certain parameters of the simulated plant or design an entirely new application. This will increase the reusability of the package over successive semesters, also giving the student to explore other systems and test their knowledge.

Attempts at developing a high performance simulator show that in order to achieve a fast yet flexible solver, utilization of a processor-like device is required. For this purpose, it may be beneficial to explore a customized processor designed to handle equations governing dynamical systems. Such a design will be able to exhibit both the flexibility of a processor and the speed of hardware-accelerated arithmetic. More detailed inspection and revised design of interface emulators is also advisable in order to decrease dependency on simulator speed and provide a more realistic interface.

As for the hybrid solution, although the capability of handling almost any plant is demonstrated, the delays in the microprocessor platform deteriorate the performance of the overall system. Methods of code optimization, as well as the use of real-time operating systems or stand-alone executables are definitely to be beneficial at this point. Exploration of distributed and/or hardware accelerated calculations is also an open end.

REFERENCES

[1] Maxwell, J. C., "On Governors," *Proceedings of the Royal Society of London*, vol. 16, no. ArticleType: primary_article / Full publication date: 1867 - 1868 / Copyright © 1867 The Royal Society, pp. 270-283, 1867.

[2] Routh, E. J., *Stability of motion*, Taylor & Francis ; Halsted Press, London : New York :, 1975.

[3] Li, Y., Ang, K., Chong, G., Feng, W., Tan, K., and Kashiwagi, H., "CAutoCSD-evolutionary search and optimisation enabled computer automated control system design," *International Journal of Automation and Computing*, vol. 1, no. 1, pp. 76-88, 2004.

[4] Sisle, M. E., and McCarthy, E. D., "Hardware-in-the-loop simulation for an active missile," *SIMULATION*, vol. 39, no. 5, pp. 159-167, 1982.

[5] Bailey, M., and Doerr, J., "Contributions of hardware-in-the-loop simulations to Navy test and evaluation," *Proceedings* pp. 33-43, Orlando, FL, USA, 1996.

[6] Cole, J. J. S., and Jolly, A. C., "Hardware-in-the-loop simulation at the U.S. Army Missile Command," *Proceedings* pp. 14-19, Orlando, FL, USA, 1996.

[7] Eguchi, H., and Yamashita, T., "Benefits of HWIL simulation to develop guidance and control systems for missiles," *Proceedings* pp. 66-73, Orlando, FL, USA, 2000.

[8] Evans, M. B., and Scholing, L. J., "The role of simulation in the development and flight test of the HiMAT vehicle," Technical Report No. H-1190; NAS 1.15:84912; NASA-TM-84912 NASA, 1984.

[9] Badaruddin, K. S., Hernandez, J. C., and Brown, J. M., "The importance of hardware-in-the-loop testing to the Cassini mission to Saturn," *Proceedings of Aerospace Conference, 2007 IEEE*, pp. 1-9, 2007.

[10] de Carufel, J., Martin, E., and Piedboeuf, J. C., "Control strategies for hardware-in-the-loop simulation of flexible space robots," *Control Theory and Applications, IEE Proceedings* -, vol. 147, no. 6, pp. 569-579, 2000.

[11] Wei, R., Jin, M. H., Xia, J. J., Xie, Z. W., Shi, J. X., and Liu, H., "High fidelity distributed hardware-in-the-loop simulation for space robot," *Proceedings of Mechatronics and Automation, Proceedings of the 2006 IEEE International Conference on*, pp. 2150-2155, 2006.

[12] Leitner, J., "A hardware-in-the-loop testbed for spacecraft formation flying applications," *Proceedings of Aerospace Conference, 2001, IEEE Proceedings.*, pp. 2/615-2/620 vol.2, 2001.

[13] Xiaofeng, W., and Vladimirova, T., "Hardware-in-loop simulation of a satellite sensor network for distributed space applications," *Proceedings of Adaptive Hardware and Systems, 2008. AHS '08. NASA/ESA Conference on*, pp. 424-431, 2008.

[14] Devie, F., and Lemaire, J., "A flexible hardware in the loop simulator for a long range autonomous underwater vehicle," *Proceedings of OCEANS '98 Conference Proceedings*, pp. 1359-1363 vol.3, 1998.

[15] Hwang, A., Seonil, Y., Tae-Yeong, K., Dae-Yong, K., Chulho, C., and Hyeonjin, C., "Verification of unmanned underwater vehicle with velocity over 10 knots guidance control system based on hardware in the loop simulation," *Proceedings of OCEANS 2009, MTS/IEEE Biloxi - Marine Technology for Our Future: Global and Local Challenges*, pp. 1-5, 2009.

[16] Krishnamurthy, P., Khorrami, F., and Ng, T. L., "Control design for unmanned sea surface vehicles: hardware-in-the-loop simulator and experimental results," *Proceedings of Intelligent Robots and Systems, 2007. IROS 2007. IEEE/RSJ International Conference on*, pp. 3660-3665, 2007.

[17] Parodi, O., Lapierre, L., and Jouvencel, B., "Hardware-in-the-loop simulators for multi-vehicles scenarios: survey on existing solutions and proposal of a new architecture," *Proceedings of Intelligent Robots and*

Systems, 2009. IROS 2009. IEEE/RSJ International Conference on, pp. 225-230, 2009.

[18] Baracos, P., Murere, G., Rabbath, C. A., and Jin, W., "Enabling PC-based HIL simulation for automotive applications," *Proceedings of Electric Machines and Drives Conference, 2001. IEMDC 2001. IEEE International*, pp. 721-729, 2001.

[19] Brennan, S., and Alleyne, A., "Using a scale testbed: Controller design and evaluation," *Control Systems Magazine, IEEE*, vol. 21, no. 3, pp. 15-26, 2001.

[20] Brennan, S., Alleyne, A., and DePoorter, M., "The Illinois Roadway Simulator-a hardware-in-the-loop testbed for vehicle dynamics and control," *Proceedings of American Control Conference, 1998. Proceedings of the 1998*, pp. 493-497 vol.1, 1998.

[21] Choi, S.-B., Lee, H., Hong, S.-R., and Cheong, C., "Control and response characteristics of a magnetorheological fluid damper for passenger vehicles," *Proceedings* pp. 438-443, Newport Beach, CA, USA, 2000.

[22] Kendall, I. R., and Jones, R. P., "An investigation into the use of hardware-in-the-loop simulation testing for automotive electronic control systems," *Control Engineering Practice*, vol. 7, no. 11, pp. 1343-1356, 1999.

[23] King, P. J., and Copp, D. G., "Hardware in the loop for automotive vehicle control systems development," *Proceedings of UKACC Control 2004 Mini Symposia*, pp. 75-78, 2004.

[24] Short, M., and Pont, M. J., "Hardware in the loop simulation of embedded automotive control system," *Proceedings of Intelligent Transportation Systems, 2005. Proceedings. 2005 IEEE*, pp. 426-431, 2005.

[25] Jae-Cheon, L., and Myuug-Won, S., "Hardware-in-the loop simulator for ABS/TCS," *Proceedings of Control Applications, 1999. Proceedings of the 1999 IEEE International Conference on*, pp. 652-657 vol. 1, 1999.

[26] Ki-Chang, L., jeong-Woo, J., Don-Ha, H., Se-Han, L., and Yong-Joo, K., "Development of antilock braking controller using hardware in-the-loop

simulation and field test," *Proceedings of Industrial Electronics Society, 2004. IECON 2004. 30th Annual Conference of IEEE*, pp. 2137-2141 Vol. 3, 2004.

[27] Boot, R., Richert, J., Schutte, H., and Rukgauer, A., "Automated test of ECUs in a hardware-in-the-loop simulation environment," *Proceedings of Computer Aided Control System Design, 1999. Proceedings of the 1999 IEEE International Symposium on*, pp. 587-594, 1999.

[28] Reorda, M. S., and Violante, M., "Hardware-in-the-loop-based dependability analysis of automotive systems," *Proceedings of On-Line Testing Symposium, 2006. IOLTS 2006. 12th IEEE International*, pp. 6 pp., 2006.

[29] Bahri, I., Naouar, M. W., Monmasson, E., Slama-Belkhodja, I., and Charaabi, L., "Design of an FPGA-based real-rime simulator for electrical systems," *Proceedings of Power Electronics and Motion Control Conference, 2008. EPE-PEMC 2008. 13th*, pp. 1365-1370, 2008.

[30] Jayalakshmi, K., and Ramanarayanan, V., "Real-time simulation of electrical machines on FPGA platform," *Proceedings of Power Electronics, 2006. IICPE 2006. India International Conference on*, pp. 259-263, 2006.

[31] Crosbie, R., Zenor, J., Bednar, R., Word, D., Hingorani, N., and Ericsen, T., "High-Speed, scalable, real-time simulation using DSP arrays," *Proceedings of Parallel and Distributed Simulation, 2004. PADS 2004. 18th Workshop on*, pp. 52-59, 2004.

[32] Karimi, S., Poure, P., and Saadate, S., "FPGA-based hardware in the loop validation for fault tolerant three-phase active filter," *Proceedings of Industrial Electronics, 2008. ISIE 2008. IEEE International Symposium on*, pp. 2189-2194, 2008.

[33] Le-Huy, P., Guerette, S., Dessaint, L. A., and Hoang, L.-H., "Dual-step real-time simulation of power electronic converters using an FPGA," *Proceedings of Industrial Electronics, 2006 IEEE International Symposium on*, pp. 1548-1553, 2006.

[34] Abourida, S., Bélanger, J., and Dufour, C., "Real-time HIL simulation of a complete PMSM drive at 10 µs time step," *Proceedings* 2005.

[35] Dufour, C., Belanger, J., Lapointe, V., and Abourida, S., "Real-time simulation on FPGA of a permanent magnet synchronous machine drive using a finite-element based model," *Proceedings of Power Electronics, Electrical Drives, Automation and Motion, 2008. SPEEDAM 2008. International Symposium on*, pp. 19-25, 2008.

[36] Bonney, J., Bowering, G., Marotz, R., and Swanson, K., "Hardware-in-the-loop emulation of mobile wireless communication environments," *Proceedings of Aerospace Conference, 2008 IEEE*, pp. 1-9, 2008.

[37] Carter, L., Dyal, J., Doshi, S., and Bagrodia, R., "A hardware-in-the-loop (HWIL) network simulator for analysis and evaluation of large-scale military wireless communication systems," *Proceedings of Military Communications Conference, 2008. MILCOM 2008. IEEE*, pp. 1-8, 2008.

[38] Clark, A. M., Kocak, D. M., Martindale, K., and Woodroffe, A., "Numerical modeling and hardware-in-the-Loop simulation of undersea networks, ocean observatories and offshore communications backbones," *Proceedings of OCEANS 2009, MTS/IEEE Biloxi - Marine Technology for Our Future: Global and Local Challenges*, pp. 1-11, 2009.

[39] Bullock, D., Johnson, B., Wells, R. B., Kyte, M., and Li, Z., "Hardware-in-the-loop simulation," *Transportation Research Part C: Emerging Technologies*, vol. 12, no. 1, pp. 73-89, 2004.

[40] Eil, K., Sangho, K., and Taek Mu, K., "Pseudo real-time evaluation of adaptive traffic control strategies using hardware-in-loop simulation," *Proceedings of Industrial Electronics Society, 2001. IECON '01. The 27th Annual Conference of the IEEE*, pp. 1910-1914 vol.3, 2001.

[41] Engelbrecht, R., "Using hardware-in-the-loop traffic simulation to evaluate traffic signal controller features," *Proceedings of Industrial Electronics Society, 2001. IECON '01. The 27th Annual Conference of the IEEE*, pp. 1920-1925 vol.3, 2001.

[42] Yan, L., Yaping, D., Lingling, M., and Liang, M., "Design and Implementation of traffic signal optimization hardware-in-loop simulation system," *Proceedings of Intelligent Computation Technology and Automation, 2009. ICICTA '09. Second International Conference on*, pp. 550-553, 2009.

[43] Grega, W., "Hardware-in-the-loop simulation and its application in control education," *Proceedings of Frontiers in Education Conference, 1999. FIE '99. 29th Annual*, pp. 12B6/7-12B612 vol.2, 1999.

[44] Shiakolas, P. S., and Piyabongkarn, D., "Development of a real-time digital control system with a hardware-in-the-loop magnetic levitation device for reinforcement of controls education," *Education, IEEE Transactions on*, vol. 46, no. 1, pp. 79-87, 2003.

[45] Tarte, Y., YangQuan, C., Wei, R., and Moore, K., "Fractional horsepower dynamometer - A general purpose hardware-in-the-loop real-time simulation platform for nonlinear control research and education," *Proceedings of Decision and Control, 2006 45th IEEE Conference on*, pp. 3912-3917, 2006.

[46] *Speedgoat GmbH*, Last accessed 02.06.2010, http://www.speedgoat.ch

[47] *dSPACE GmbH*, Last accessed 05.06.2010, http://www.dspaceinc.com/ww/en/inc/home.cfm

[48] *Opal-RT Technologies Inc.*, Last accessed 02.06.2010, http://www.opal-rt.com

[49] *Applied Dynamics International Inc.*, Last accessed 02.06.2010, http://www.adi.com

[50] *National Instruments Inc. - Hardware-in-the-Loop Testing*, Last accessed 05.06.2010, http://www.ni.com/hil/

[51] *The MathWorks - xPC Target*, Last accessed 05.06.2010, http://www.mathworks.com/products/xpctarget/

[52] Reynolds, O., "On the theory of lubrication and its application to Mr. Beauchamp Tower's experiments, including and experimental determination

of the viscosity of olive oil," *Philosophical Transactions of the Royal Society*, vol. 177, pp. 157-234, 1886.

[53] Stribeck, R., "Die wesentlichen Eigenschaften der Gleit- und Rollenlager," *Zeitschrift des Vereines Deutscher Ingenieure*, vol. 46, no. 38, 39, pp. 1342-48, 1432-37, 1902.

[54] Morin, A. J., "New friction experiments carried out at Metz in 1831-1833," *Proceedings of the French Royal Academy of Sciences*, vol. 4, pp. 1-128, 1833.

[55] Rabinowicz, E., "The nature of the static and kinetic coefficients of friction," *Journal of Applied Physics*, vol. 22, no. 11, pp. 1373-79, 1951.

[56] Johannes, V. I., Green, M. A., and Brockley, C. A., "The role of the rate of application of the tangential force in determining the static friction coefficient," *Wear*, vol. 24, pp. 381-385, 1973.

[57] Richardson, R. S. H., and Nolle, H., "Surface friction under time-dependent loads," *Wear*, vol. 37, no. 1, pp. 87-101, 1976.

[58] Courtney-Pratt, J., and Eisner, E., "The effect of a tangential force on the contact of metallic bodies," *Proceedings of the Royal Society of London*, vol. A238, pp. 529-550, 1957.

[59] Karnopp, D., "Computer Simulation of Stick-Slip Friction in Mechanical Dynamic Systems," *Journal of Dynamic Systems, Measurement, and Control*, vol. 107, no. 1, pp. 100-103, 1985.

[60] Armstrong-Helouvry, B., Dupont, P., and Canudas de Wit, C., "A survey of models, analysis tools and compensation methods for the control of machines with friction," *Automatica*, vol. 30, no. 7, pp. 1083-1138, 1994.

[61] Dahl, P., "A solid friction model," Technical Report No. TOR-0158(3107-18)-1, The Aerospace Corporation, El Segundo, CA, 1968.

[62] Bliman, P.-A., "Mathematical study of the Dahl's friction model," *European Journal of Mechnics. A / Solids*, vol. 11, no. 6, pp. 835-848, 1992.

[63] Haessig, D. A., and Friedland, B., "On the modeling and simulation of friction," *Journal of Dynamic Systems, Measurement and Control*, vol. 113, no. 3, pp. 354-362, 1991.

[64] Bliman, P.-A., and Sorine, M., "Friction modeling by hysteresis operators: Application to Dahl, stiction and Stribeck effects.," *Proceedings of Models of Hysteresis*, Trento, Italy, 1991.

[65] Bliman, P.-A., and Sorine, M., "A system-theoretic approach of systems with hysteresis: Application to friction modeling and compensation," *Proceedings of Second European Control Conference*, pp. 1844-49, Groningen, The Netherlands, 1993.

[66] Bliman, P.-A., and Sorine, M., "Easy-to-use realistic dry friction models for automatic control," *Proceedings of Third European Control Conference*, pp. 3788-3794, Rome, Italy, 1995.

[67] Canudas de Wit, C., Olsson, H., Åström, K. J., and Lischinsky, P., "A new model for control of systems with friction," *Automatic Control, IEEE Transactions on*, vol. 40, no. 3, pp. 419-425, 1995.

[68] Reynolds, O., "Creep theory of belt drive mechanics," *The Engineer*, vol. 38, no. 396, 1847.

[69] Swift, H. W., "Power transmission by belts: An investigation of fundamentals," *Proceedings of the Institution of Mechanical Engineers*, vol. 2, no. 659, 1928.

[70] Abrate, S., "Vibrations of belts and belt drives," *Mechanism and Machine Theory*, vol. 27, no. 6, pp. 645-659, 1992.

[71] Hace, A., Jezernik, K., and Sabanovic, A., "SMC with disturbance observer for a linear nelt drive," *Industrial Electronics, IEEE Transactions on*, vol. 54, no. 6, pp. 3402-3412, 2007.

[72] Bechtel, S. E., Vohra, S., Jacob, K. I., and Carlson, C. D., "The Stretching and Slipping of Belts and Fibers on Pulleys," *Journal of Applied Mechanics*, vol. 67, no. 1, pp. 197-206, 2000.

[73] Rubin, M. B., "An exact solution for steady motion of an extensible eelt in multipulley belt drive systems," *Journal of Mechanical Design*, vol. 122, no. 3, pp. 311-316, 2000.

[74] Firbank, T. C., "Mechanics of the Belt Drive," *International Journal of Mechanical Science*, vol. 12, no. 12, 1970.

[75] Childs, T. H., and Parker, J. E., "Power transmission by flat, V and timing belts," *Proceedings of 15th Leeds-Lyon Symposium on Tribology*, pp. 133-142, 1989.

[76] Alciatore, D. G., and Traver, A. E., "Multipulley belt drive mechanics: Creep theory vs shear theory," *Journal of Mechanical Design*, vol. 117, no. 4, pp. 506-511, 1995.

[77] Gerbert, G., "Belt Slip---A Unified Approach," *Journal of Mechanical Design*, vol. 118, no. 3, pp. 432-438, 1996.

[78] Gerbert, G., "On flat belt slip," *Proceedings of 17th Leeds-Lyon Symposium on Tribology*, pp. 333-339, 1996.

[79] Slotine, J. J. E., and Weiping, L., *Applied Nonlinear Control*, Prentice-Hall, Englewood Cliff, New Jersey, 1991.

[80] Sarkar, N., Ellis, R. E., and Moore, T. N., "Backlash detection in geared mechanisms: Modeling, simulation and experimentation," *Mechanical Systems and Signal Processing*, vol. 11, pp. 391-408, 1997.

[81] Nordin, M., and Gutman, P.-O., "Controlling mechanical systems with backlash--a survey," *Automatica*, vol. 38, no. 10, pp. 1633-1649, 2002.

[82] Yamamura, S., and Nakagawa, T., "Transient phenomena and control of ac servomotor-proposal of field acceleration method," *Electrical Engineering in Japan*, vol. 101, no. 5, pp. 69-75, 1981.

[83] Depenbrock, M., "Direct self-control (DSC) of inverter-fed induction machine," *Power Electronics, IEEE Transactions on*, vol. 3, no. 4, pp. 420-429, 1988.

[84] French, C., and Acarnley, P., "Direct torque control of permanent magnet drives," *Industry Applications, IEEE Transactions on*, vol. 32, no. 5, pp. 1080-1088, 1996.

[85] Mir, S., Elbuluk, M. E., and Zinger, D. S., "PI and fuzzy estimators for tuning the stator resistance in direct torque control of induction machines," *Proceedings of Power Electronics Specialists Conference, PESC '94 Record., 25th Annual IEEE*, pp. 744-751 vol.1, 1994.

[86] Behera, R., and Das, S., "Improved direct torque control of induction motor with dither injection," *Sadhana*, vol. 33, no. 5, pp. 551-564, 2008.

[87] Soong, W. L., and Miller, T. J. E., "Field-weakening performance of brushless synchronous AC motor drives," *Electric Power Applications, IEE Proceedings* -, vol. 141, no. 6, pp. 331-340, 1994.

[88] Moore, S., and Ehsani, M., "Effect on vehicle performance of extending the constant power region of electric drive motors," *Proceedings of International Congress & Exposition*, Detroit, MI, USA, 1999.

[89] Wright, D., *Notes on Design and Analysis of Machine Elements*, Last accessed 25.06.2010, http://school.mech.uwa.edu.au/~dwright/DANotes/

[90] Grimheden, M., and Törngren, M., "How should embedded systems be taught?: Experiences and snapshots from Swedish higher engineering education," *SIGBED Rev.*, vol. 2, no. 4, pp. 34-39, 2005.

[91] *Microsoft XNA*, Last accessed http://www.xna.com/

[92] Yen-Ju, L., Chen-Tung, L., Chi-Feng, W., Shih-Arn, H., and Ying-Hsi, L., "Microprocessor modeling and simulation with SystemC," *Proceedings of VLSI Design, Automation and Test, 2007. VLSI-DAT 2007. International Symposium on*, pp. 1-4, 2007.

[93] Gorton, I., Kerridge, J., and Jervis, B., "Simulating microprocessor systems using occam and a network of transputers," *Computers and Digital Techniques, IEE Proceedings E*, vol. 136, no. 1, pp. 22-28, 1989.

[94] Munson, B. R., Young, D. F., and Okiishi, T. H., *Fundamentals of Fluid Mechanics*, 3rd Ed., John Wiley, New York, 1998.

[95] Peterka, J. A., Tan, Z., Bienkiwicz, B., and Cermak, J. E., "Wind loads on heliostats and parabolic dish collectors," Colorado State University, Fort Collins, Colorado, 1988.

[96] *Custom Computer Services Inc.*, Last accessed 03.06.2010, http://www.ccsinfo.com

[97] Özsipahi, M., "ME534 Final project report: Design of a satellite-antenna tracking controller," 2010.

[98] Usenmez, S., Dilan, R. A., Yaman, U., Mutlu, B. R., Dolen, M., and Koku, A. B., "Çevrimiçi donanım benzetimi için yeni bir yazılım paketi: Cadmus," *Proceedings of Turkish National Committee of Automatic Control*, 2008.

[99] *Woorim t&i Terasic Altera DE1 Product Page*, Last accessed 24.06.2010, http://www.woorimtni.co.kr/terasic/terasic_fpga_06_view.jsp

[100] Usselmann, R., "Open Floating Point Unit Manual," 2005.

[101] Usenmez, S., Dilan, R. A., Dolen, M., and Koku, A. B., "Bir doğru akım motorunun FPGA üzerinde gerçek zamanlı benzetiminin gerçekleştirilmesi," *Proceedings of Turkish National Committee of Automatic Control*, 2009.

[102] Usenmez, S., Dilan, R. A., Dolen, M., and Koku, A. B., "Real-time hardware-in-the-loop simulation of electrical machine systems using FPGAs," *Proceedings of Electrical Machines and Systems, 2009. ICEMS 2009. International Conference on*, pp. 1-6, Tokyo, Japan, 2009.

[103] *Atmel AT32AP7000 AVR32 32-bit Microcontroller Datasheet*, Last accessed 29.05.2010, http://www.atmel.com/dyn/resources/prod_documents/32003S.pdf

[104] GE-Fanuc, "α Series AC Servo Motor Descriptions Manual," 1995.

[105] *NPlot Charting Library Project Page*, Last accessed http://live.xbox.com/en-GB/profile/profile.aspx?pp=0&GamerTag=ilkekaya

[106] Corbet, J., Rubini, A., and Kroah-Hartman, G., *Linux Device Drivers*, Third Ed., O'Reilly Media, 2005.

[107] *AVR32 Studio Product Page*, Last accessed http://www.atmel.com/dyn/products/tools_card.asp?tool_id=4116

[108] *Eclipse Foundation Home Page*, Last accessed http://www.eclipse.org/

[109] *Atmel Buildroot for AVR32 Home Page*, Last accessed http://www.atmel.no/buildroot/

[110] Krosgaard, P., *Buildroot Project Page*, Last accessed http://buildroot.uclibc.org/

[111] *Microchip Inc. PIC18F4520 Product Page*, Last accessed 03.06.2010, http://www.microchip.com/wwwproducts/Devices.aspx?dDocName=en010297

[112] *Microchip PIC18F4520 Datasheet*, Last accessed 03.06.2010, http://ww1.microchip.com/downloads/en/DeviceDoc/39631E.pdf

[113] *AVRfreaks Wiki NGW/NGW100 Features Page*, Last accessed 03.06.2010, http://www.avrfreaks.net/wiki/index.php/Documentation:NGW/NGW100_Features

[114] *Altera DE1 Development and Education Board Product Page*, Last accessed 29.05.2010, http://university.altera.com/materials/boards/de1/?board=DE1

[115] *Altera Cyclone II Device Family Datasheet*, Last accessed 29.05.2010, http://www.altera.com/literature/hb/cyc2/cyc2_cii5v1_01.pdf

[116] *Altera Nios II Embedded Evaluation Kit Product Page*, Last accessed 29.05.2010, http://www.altera.com/products/devkits/altera/kit-cyc3-embedded.html

[117] *Altera Cyclone III Device Handbook*, Last accessed 29.05.2010, http://www.altera.com/literature/hb/cyc3/cyc3_ciii5v1.pdf

APPENDIX A

HARDWARE SPECIFICATIONS

A.1 PIC Development Board and Microchip PIC Microcontrollers

This section provides specifications for the PIC Development Board and its components. The board itself accommodates:

- Microchip PIC18F4520 microcontroller
- 20 MHz crystal oscillator
- Two 0-20kΩ potentiometers (connected to pins configurable as D/A converter)
- Push button (connected to pin configurable as external interrupt)
- Three LEDs
- Serial port with MAX232 TTL – RS-232 converter IC
- Female sockets connected to all unused pins on the microcontroller (5V and ground wires also provided)
- Reset push button
- On-off switch

Table A.1: Microchip PIC18F4520 microcontroller specifications [111-112]

Operating Frequency	40 MHz
CPU Speed	10 MIPS
SRAM Memory	1,536
Program Memory	16,384
EEPROM Memory	256
Interrupts	20
I/O Pins	36
Timers	1 × 8-bit, 3 × 16-bit
A/D Converters	13 × 10-bit channels
Capture/Compare/PWM	1 × CCP, 1 × ECCP modules
Serial Communications	1 × SSP (SPI/I^2C), 1 × EUSART
Parallel Communications	PSP
Analog Comparators	2

A.2 NGW100 Network Gateway Kit and AT32AP7000 Microprocessor

This section provides specifications for the NGW100 Network Gateway Kit and its components. The NGW100 itself accommodates [113]:

- Atmel AT32AP7000 CPU
- 32 MB SDRAM
- 16 MB on-board flash
- SD-Card/MMC slot
- ATtiny24 board controller & ISP interface for board identification
- JTAG & Nexus programming/debugging interface
- RS-232 port
- Two Ethernet ports
- USB port
- 63 expansion pins for general purpose I/O and AT32AP7000 peripheral modules
- Power & status LEDs
- Two user controllable LEDs

- Reset push button
- Boot select jumper

The specifications of the AT32AP7000 microprocessor are [103]:

- AVR32 32-bit Microcontroller
    - 210 DMIPS throughput at 150 MHz
    - 16 KB instruction cache and 16 KB data cache
    - MMU
    - Single-cycle RISC instruction set including SIMD and DSP instructions
    - Java hardware acceleration
- Pixel coprocessor for video acceleration
- Multi-hierarchy bus system
- 32 KB SRAM data memory
- External SDRAM, DataFlash, SRAM, MMC, SD, Compact Flash, Smart Media, NAND Flash memory interfaces
- DMA Controller
- Interrupt Controller
- System Functions
    - Power and clock manager
    - Crystal oscillator with PLL
    - Watchdog timer
    - Real-time clock
- 6 × multifunction timer/counters
    - Three external clock inputs, I/O pins, PWM, capture and various counting capabilities
- 4 × USART
    - 115.2 kbps IrDA Modulation and Demodulation

- Hardware and software handshaking
- 3 × SSP controllers
  - Supports I^2S, SPI and generic frame-based protocols
- Two-wire interface
  - Sequential read/write operations, I^2C compliant
- LCD interface
  - Supports TFT displays
  - Configurable pixel resolution supporting QCIF/QVGA/VGA/SVGA configurations
- 12-bit image sensor interface for CMOS cameras
- USB 2.0 high speed (480 Mbps) device
- 2 × 802.3 Ethernet MAC 10/100 Mbps interfaces
  - Supports MII and RMII
- 16-bit stereo audio bit stream DAC with sample rates up to 50 kHz
- On-chip debug system
  - Nexus class 3
  - Full speed, non-intrusive data and program trace
  - Runtime control and JTAG interface
- Package/Pins
  - AT32AP7000: 256-ball CTBGA 1.0 mm pitch/160 GPIO pins
- Power supplies
  - 1.65V to1.95V VDDCORE
  - 3.0V to 3.6V VDDIO

A.3 Terasic Altera DE1 Cyclone II FPGA Starter Kit

This section provides specifications for the DE1 Cyclone II FPGA Starter Kit and its components. The DE1 itself accommodates [114]:

- Altera Cyclone II EP2C20F484C7 FPGA Chip

- Built-in USB-Blaster chip for FPGA configuration
- 8 MB SDRAM
- 512 kB SRAM
- 4 MB flash memory
- 10 × toggle switches
- 4 × push buttons
- 18 × LEDs
- 4 × 7-segment digit display
- 50 MHz and 27 MHz clock sources, SMA external clock input
- 24-bit audio codec chip with line in, line out and microphone in jacks
- VGA video output with 4-bit resistor array DAC
- RS-232 serial port
- PS/2 mouse/keyboard port
- 2 × 40-pin expansion headers
- SD-Card slot

Table A.2: Specifications of Cyclone II EP2C20F484C7 FPGA chip [115]

Logic Elements	18,752
M4K RAM blocks (4 Kbits + 512 parity bits)	52
Total RAM bits	239,616
Embedded multipliers (9-bit)	52
PLLs	4
User I/O pins	315

A.4 Terasic Altera Nios II Embedded Evaluation Kit

This section provides specifications for the Nios II Embedded Evaluation Kit and its components. The boards making up the kit accommodate [116]:

- Altera Cyclone II EP3C25F324 FPGA Chip
- Built-in USB-Blaster chip for FPGA configuration
- 32 MB DDR SDRAM

- 1 MB SSRAM
- 16 MB flash memory
- 4 × push buttons
- 4 × LEDs
- 50 MHz clock source
- 24-bit audio codec chip with line in, line out and microphone in jacks
- VGA video output with 4-bit resistor array DAC
- Composite TV-in
- RS-232 serial port
- PS/2 mouse/keyboard port
- Ethernet connector
- JTAG connector
- SD-Card slot

Table A.3: Specifications of Cyclone III EP3C25F324 FPGA chip [117]

Logic Elements	26,624
M9K RAM blocks (9 Kbits)	66
Total RAM bits	608,256
Embedded multipliers (9-bit)	132
PLLs	4
User I/O pins	215

APPENDIX B

SOURCE CODE LISTINGS

B.1 Introduction

The listing of source code developed during the thesis studies is presented in this appendix for completeness. Table B.1 provides a list of the included source files, while the following sections document certain important sections of these files.

Table B.1: Source files developed for the applications

File name	Lang.	Description	Related Chapter
MainForm.cs	C#	Main application and GUI routines	4
Antenna.cs	C#	Equations governing antenna behavior	4
Environment.cs	C#	Variables regarding wind properties	4
Satellite.cs	C#	Equations governing satellite motion	4
Simulation.cs	C#	Primary simulation routines	4
StateSolver.cs	C#	4th order Runge-Kutta solver routine	4
Protoc16.h	C	16-bit communication protocol header	4
PWM_Receive.v	Verilog	Modified (synced) PWM receiver	5, 6
Encoder.vhd	VHDL	Time-scalable encoder emulator	5, 6
main.cpp	C++	Main simulation process	6
CadmusDevice.h, .cpp	C++	FPGA board management class	6
CadmusManager.h, .cpp	C++	PC communication management class	6
DisturbanceSource.h, .cpp	C++	Generated/pre-recorded disturbance provider	6
EncoderCommander.h, .cpp	C++	Command generator for encoder emulator	6
Solver_RK4.h, .cpp	C++	4th order Runge-Kutta solver routine	6
NGW_Interface.vhd	VHDL	FPGA manager and NGW100 interface	6
InputSignaller.vhd	VHDL	Multiple input command detection module	6
hils_milling_distgen.m	m-file	Machining center disturbance generation	6

B.2 Source code – MainForm.cs

```
using System;
using System.Collections.Generic;
using System.Drawing;
using System.IO;
using System.Windows.Forms;

namespace SatelliteAntenna
{
        public partial class MainForm : Form
        {
                public MainForm()
                {
                        InitializeComponent();
                }

                // On launch
                private void Form1_Load(object sender, EventArgs e)
                {
                        // Setup scopes on view tab - grids, colors etc.
                        #region Scopes
                        // ...
                        #endregion

                        // Print parameters in settings tab for user convenience
                        #region System Parameters
                        // ...
                        #endregion

                        // Enumerate serial ports on the computer
                        string[] PortNames =
System.IO.Ports.SerialPort.GetPortNames();
                        if (PortNames.Length > 0)
                                cmbSettings_SerialComm_Port.Items.AddRange(PortNames);
                        else
                                MessageBox.Show("No COM ports were found. Please check
your hardware connections and settings.", "Error", MessageBoxButtons.OK,
MessageBoxIcon.Error);

                        // Reload settings from the previous run of the program,
                        // the user won't have to re-adjust everything
                        bool SettingsLoaded = false;
                        if (File.Exists("Settings.dat"))
                        {
                                try
                                {
                                        // Read and apply settings from file
                                        // ...

                                        SettingsLoaded = true;
                                }
                                catch
                                {
                                        // Corrupted settings file, delete it
                                        File.Delete("Settings.dat");
                                }
                        }

                        // Default settings if no record exists or it exists but is
                        // corrupted
                        if (!SettingsLoaded)
                        {
                                // ...
                        }

                        Initialized = true;
                }
```

```
		// User changes sampling time
		private void
cmbSettings_Simulation_SamplingTime_SelectedIndexChanged(object sender, EventArgs
e)
		{
			// Assign sampling time ts
			int SamplingTime =
Convert.ToInt32(cmbSettings_Simulation_SamplingTime.Text);
			Simulation.SamplingTime = (float)SamplingTime / 1000;

			// Populate data recording intervals, starting at ts/2
			// down to 0.25ms
			List<int> Items = new List<int>();
			for (int i = SamplingTime - 1; i > 0; i--)
				if (((SamplingTime / i) * i) == SamplingTime)
					Items.Add(i);

			cmbSettings_Simulation_LogInterval.Items.Clear();
			foreach (int i in Items)
				cmbSettings_Simulation_LogInterval.Items.Add(i);

			cmbSettings_Simulation_LogInterval.Items.Add(0.5);
			cmbSettings_Simulation_LogInterval.Items.Add(0.25);

			cmbSettings_Simulation_LogInterval.SelectedIndex = 0;
		}

		// At program exit
		private void MainForm_FormClosing(object sender, FormClosingEventArgs
e)
		{
			Simulation.Stop();

			// Record application settings
			StreamWriter Writer = new
System.IO.StreamWriter("Settings.dat", false);
			Writer.WriteLine("// Do not modify the contents of this file
manually. Delete the file if you wish to reset your settings. #21");

			// ...
			// ...
			// ...

			Writer.Flush();
			Writer.Close();
		}

		// Record mouse button press for adjusting antenna angles
		// Only works when simulation is not running
		private void satelliteGraphicsDeviceControl_MouseDown(object sender,
MouseEventArgs e)
		{
			if (Simulation.State == RunState.Stopped)
			{
				if (e.Button == MouseButtons.Left)
				{
					m_CursorPosition = Cursor.Position;
					Cursor = Cursors.SizeAll;
				}
			}
		}

		// Rotate antenna when the user drags the mouse
		// This is for setting intial position
		private void satelliteGraphicsDeviceControl_MouseMove(object sender,
MouseEventArgs e)
		{
			if (!m_CursorPosition.IsEmpty)
			{
```

```
                              Program.Antenna.Azimuth +=
Angle.ToRadian(Cursor.Position.X - m_CursorPosition.X) /
Microsoft.Xna.Framework.MathHelper.Pi;

                              float newElevation = Program.Antenna.Elevation -
Angle.ToRadian(Cursor.Position.Y - m_CursorPosition.Y) /
Microsoft.Xna.Framework.MathHelper.Pi;
                              newElevation =
Microsoft.Xna.Framework.MathHelper.Clamp(newElevation, 0f,
Microsoft.Xna.Framework.MathHelper.Pi);

                              Program.Antenna.Elevation = newElevation;

                              m_CursorPosition = Cursor.Position;
                       }
               }

               // Record mouse button release after adjusting antenna angles
               private void satelliteGraphicsDeviceControl_MouseUp(object sender,
MouseEventArgs e)
               {
                       m_CursorPosition = Point.Empty;
                       Cursor = Cursors.Default;
               }

               private bool Initialized = false;

               private Point m_CursorPosition = Point.Empty;
       }
}
```

B.3 Source code – Antenna.cs

```
using System;
using Microsoft.Xna.Framework;

namespace SatelliteAntenna
{
       public class Antenna
       {
               public Antenna()
               {
                       m_Latitude = 0; // float.ToRadian(39.892365f); //
39°53'32.51"N
                       m_Longitude = 0; // float.ToRadian(32.783357f); //
32°47'0.09"E
               }

               public void Think()
               {
                       float x1, x2, dx;
                       float[] States;

                       #region Wind

                       float CosElevation = (float)Math.Cos(Elevation);

                       Vector3 Facing = new Vector3();
                       Facing.X = (float)Math.Cos(Azimuth) * CosElevation;
                       Facing.Y = (float)Math.Sin(Azimuth) * CosElevation;
                       Facing.Z = (float)Math.Sin(Elevation);

                       float CosGamma = Vector3.Dot(Facing,
Program.Environment.WindDirection);
                       float Gamma = (float)Math.Acos(CosGamma);
```

```
                        float Area = (0.55f + (0.45f * Math.Abs(CosGamma))) *
0.636173f;
                        float DragCoefficient;
                        if (CosGamma < 0)
                                DragCoefficient = 0.4f + 0.51f * (1 +
(float)Math.Cos(2 * Gamma));
                        else
                                DragCoefficient = 0.4f + 0.275f * (1 +
(float)Math.Cos(2 * Gamma));

                        float Force = 0.602f * Program.Environment.WindSpeed *
Program.Environment.WindSpeed * DragCoefficient * Area;

                        m_Azimuth_WindTorque = Force *
(float)Math.Sin(Program.Environment.WindYaw - Azimuth);
                        m_Elevation_WindTorque = Force *
(float)Math.Sin(Program.Environment.WindPitch - Elevation);

                        #endregion

                        #region Azimuth

                        if (!Program.MainForm.rbMain_ControlAxes_Elevation.Checked)
                        {
                                // Torque capability
                                if (Math.Abs(m_Azimuth_Omega1) <=
m_Azimuth_MaximumSpeed)
                                        m_Azimuth_Torque1A = m_Azimuth_TorqueCommand;
                                else
                                        m_Azimuth_Torque1A = m_Azimuth_TorqueCommand *
m_Azimuth_MaximumSpeed / Math.Abs(m_Azimuth_Omega1);

                                // Backlash
                                x1 = m_Azimuth_Theta1;
                                x2 = m_Azimuth_Theta2 * m_Azimuth_Radius2 /
m_Azimuth_Radius1;
                                dx = x1 + x2;

                                m_Azimuth_Torque1B = StateEquation_g(dx,
m_Azimuth_Backlash, m_Azimuth_TorsionConstant);
                                m_Azimuth_Torque2 = -(m_Azimuth_Radius2 /
m_Azimuth_Radius1) * m_Azimuth_Torque1B;

                                States = new float[] { m_Azimuth_Theta1,
m_Azimuth_Omega1, m_Azimuth_Theta2, m_Azimuth_Omega2 };

                                // Apply RK4
                                States =
StateSolver.RungeKutta4(StateEquation_Azimuth, States, new float[] {
m_Azimuth_Torque1A, m_Azimuth_Torque1B, m_Azimuth_Torque2, m_Azimuth_WindTorque },
Simulation.Time, Simulation.Step);

                                m_Azimuth_Theta1 = States[0];
                                m_Azimuth_Omega1 = States[1];
                                m_Azimuth_Theta2 = States[2];
                                m_Azimuth_Omega2 = States[3];
                        }
                        else
                        {
                                // No control will be done for azimuth
                                m_Azimuth_Omega1 = 0;
                                m_Azimuth_Omega2 = 0;

                                Azimuth = Program.Satellite.Azimuth;
                        }

                        #endregion

                        #region Elevation
                        if (!Program.MainForm.rbMain_ControlAxes_Azimuth.Checked)
                        {
                                // Torque capability
                                if (Math.Abs(m_Elevation_Omega1) <=
m_Elevation_MaximumSpeed)
                                        m_Elevation_Torque1A =
m_Elevation_TorqueCommand;
                                else
                                        m_Elevation_Torque1A =
m_Elevation_TorqueCommand * m_Elevation_MaximumSpeed /
Math.Abs(m_Elevation_Omega1);

                                // Backlash
                                x1 = m_Elevation_Theta1;
                                x2 = m_Elevation_Theta2 * m_Elevation_Radius2 /
m_Elevation_Radius1;
                                dx = x1 + x2;

                                m_Elevation_Torque1B = StateEquation_g(dx,
m_Elevation_Backlash, m_Elevation_TorsionConstant);
                                m_Elevation_Torque2 = -(m_Elevation_Radius2 /
m_Elevation_Radius1) * m_Elevation_Torque1B;
                                float ConstraintTorque =
StateEquation_ElevationConstraintTorque();

                                States = new float[] { m_Elevation_Theta1,
m_Elevation_Omega1, m_Elevation_Theta2, m_Elevation_Omega2 };

                                // Apply RK4
                                States =
StateSolver.RungeKutta4(StateEquation_Elevation, States, new float[] {
m_Elevation_Torque1A, m_Elevation_Torque1B, m_Elevation_Torque2,
m_Elevation_WindTorque, ConstraintTorque }, Simulation.Time, Simulation.Step);

                                m_Elevation_Theta1 = States[0];
                                m_Elevation_Omega1 = States[1];
                                m_Elevation_Theta2 = States[2];
                                m_Elevation_Omega2 = States[3];
                        }
                        else
                        {
                                // No control will be done for elevation
                                m_Elevation_Omega1 = 0;
                                m_Elevation_Omega2 = 0;

                                Elevation = Program.Satellite.Elevation;
                        }

                        #endregion

                        #region Signal

                        float AzimuthError = Program.Satellite.Azimuth - Azimuth;
                        while (AzimuthError > Mathematics.PI)
                                AzimuthError -= Mathematics.TwoPI;
                        while (AzimuthError < -Mathematics.PI)
                                AzimuthError += Mathematics.TwoPI;

                        float ElevationError = Program.Satellite.Elevation -
Elevation;

                        AzimuthError = AzimuthError * AzimuthError;
                        ElevationError = ElevationError * ElevationError;

                        m_SignalStrength = 10 * (float)Math.Exp(-625 * (AzimuthError
+ ElevationError));

                        #endregion
```

```
		}

		private float[] StateEquation_Azimuth(float[] X, float[] U, float
Time)
		{
			float[] Result = new float[X.Length];

			float NetTorque1 = U[0] - U[1];
			if (m_Azimuth_Omega1 > 0)
				NetTorque1 -= m_Azimuth_CoulombFriction1;
			else if (m_Azimuth_Omega1 < 0)
				NetTorque1 += m_Azimuth_CoulombFriction1;
			else
			{
				// Static friction
				if (Math.Abs(NetTorque1) < m_Azimuth_CoulombFriction1)
					NetTorque1 = 0f;
				else
				{
					if (NetTorque1 > 0)
						NetTorque1 -=
m_Azimuth_CoulombFriction1;
					else
						NetTorque1 +=
m_Azimuth_CoulombFriction1;
				}
			}

			float NetTorque2 = U[2] + U[3];
			if (m_Azimuth_Omega2 > 0)
				NetTorque2 -= m_Azimuth_CoulombFriction2;
			else if (m_Azimuth_Omega2 < 0)
				NetTorque2 += m_Azimuth_CoulombFriction2;
			else
			{
				// Static friction
				if (Math.Abs(NetTorque2) < m_Azimuth_CoulombFriction2)
					NetTorque2 = 0f;
				else
				{
					if (NetTorque2 > 0)
						NetTorque2 -=
m_Azimuth_CoulombFriction2;
					else
						NetTorque2 +=
m_Azimuth_CoulombFriction2;
				}
			}

			Result[0] = X[1];
			Result[1] = -(m_Azimuth_ViscousFriction1 /
m_Azimuth_Inertia1) * X[1] + (1 / m_Azimuth_Inertia1) * (NetTorque1);
			Result[2] = X[3];
			Result[3] = -(m_Azimuth_ViscousFriction2 /
m_Azimuth_Inertia2) * X[3] + (1 / m_Azimuth_Inertia2) * (NetTorque2);

			return Result;
		}

		private float[] StateEquation_Elevation(float[] X, float[] U, float
Time)
		{
			float[] Result = new float[X.Length];

			float NetTorque1 = U[0] - U[1];

			if (m_Elevation_Omega1 > 0)
				NetTorque1 -= m_Elevation_CoulombFriction1;
			else if (m_Elevation_Omega1 < 0)
```

```
                NetTorque1 += m_Elevation_CoulombFriction1;
            else
            {
                // Static friction
                if (Math.Abs(NetTorque1) <
m_Elevation_CoulombFriction1)
                    NetTorque1 = 0f;
                else
                {
                    if (NetTorque1 > 0)
                        NetTorque1 -=
m_Elevation_CoulombFriction1;
                    else
                        NetTorque1 +=
m_Elevation_CoulombFriction1;
                }
            }

            float NetTorque2 = U[2] + U[3] + U[4];
            if (m_Elevation_Omega2 > 0)
                NetTorque2 -= m_Elevation_CoulombFriction2;
            else if (m_Elevation_Omega2 < 0)
                NetTorque2 += m_Elevation_CoulombFriction2;
            else
            {
                // Static friction
                if (Math.Abs(NetTorque2) <
m_Elevation_CoulombFriction2)
                    NetTorque2 = 0f;
                else
                {
                    if (NetTorque2 > 0)
                        NetTorque2 -=
m_Elevation_CoulombFriction2;
                    else
                        NetTorque2 +=
m_Elevation_CoulombFriction2;
                }
            }

            Result[0] = X[1];
            Result[1] = -(m_Elevation_ViscousFriction1 /
m_Elevation_Inertia1) * X[1] + (1 / m_Elevation_Inertia1) * (NetTorque1);
            Result[2] = X[3];
            Result[3] = -(m_Elevation_ViscousFriction2 /
m_Elevation_Inertia2) * X[3] + (1 / m_Elevation_Inertia2) * (NetTorque2);

            return Result;
        }

        private float StateEquation_g(float Difference, float Backlash, float
Constant)
        {
            float HalfLash = Backlash / 2;

            if (Difference >= HalfLash)
                return ((Difference - HalfLash) * Constant);
            else if (Difference <= -HalfLash)
                return ((Difference + HalfLash) * Constant);
            else
                return 0;
        }

        private float StateEquation_ElevationConstraintTorque()
        {
            if (Elevation < 0)
                return Elevation * -2500;
            else if (Elevation > Math.PI)
                return ((float)Math.PI - Elevation) * 2500;
```

```
                        else
                                return 0;
                }

                public float SignalStrength { get { return m_SignalStrength; } }

                // Outputs
                private float m_SignalStrength;

                // Inputs
                private float m_Azimuth_TorqueCommand;
                private float m_Elevation_TorqueCommand;

                // States
                private float m_Azimuth_Theta1;
                public float m_Azimuth_Omega1;
                private float m_Azimuth_Theta2;
                private float m_Azimuth_Omega2;

                private float m_Elevation_Theta1;
                private float m_Elevation_Omega1;
                private float m_Elevation_Theta2;
                private float m_Elevation_Omega2;

                // Internals
                private float m_Azimuth_Torque1A;
                private float m_Azimuth_Torque1B;
                private float m_Azimuth_Torque2;
                private float m_Azimuth_WindTorque;

                private float m_Elevation_Torque1A;
                private float m_Elevation_Torque1B;
                private float m_Elevation_Torque2;
                private float m_Elevation_WindTorque;

                // Parameters
                public const float m_Elevation_Radius1 = 0.005f;
                                // m
                public const float m_Elevation_Radius2 = 0.150f;
                                // m
                public const float m_Elevation_GearRatio = m_Elevation_Radius2 /
m_Elevation_Radius1;

                public const float m_Elevation_InertiaTotal = 0.00015f;

                        // kg * m^2
                public const float m_Elevation_Inertia1 = m_Elevation_InertiaTotal /
2f;                                                                                             //
kg * m^2
                public const float m_Elevation_Inertia2 = m_Elevation_Inertia1 *
(m_Elevation_GearRatio * m_Elevation_GearRatio);    // kg * m^2

                public const float m_Elevation_ViscousFriction2 = 0.003f;

                                                        // N * m / (rad / s)
                public const float m_Elevation_ViscousFriction1 =
m_Elevation_ViscousFriction2 * ((m_Elevation_GearRatio - 1) /
m_Elevation_GearRatio);       // N * m / (rad / s)

                public const float m_Elevation_CoulombFriction1 = 0.1f;    // N * m
                public const float m_Elevation_CoulombFriction2 = 18f;     // N * m

                public const float m_Elevation_MaximumTorque = 30f;
                                        // N * m
                public const float m_Elevation_MaximumSpeed = (1200f / 60f) *
MathHelper.TwoPi;      // rad / s
                public const float m_Elevation_TorsionConstant = 10f;
                                        // N * m / rad
```

```
		private float m_Elevation_Backlash = 0.000070f;
						// rad

		public const float m_Azimuth_Radius1 = 0.005f;
			// m
		public const float m_Azimuth_Radius2 = 0.150f;
			// m
		public const float m_Azimuth_GearRatio = m_Azimuth_Radius2 /
m_Azimuth_Radius1;

		public const float m_Azimuth_InertiaTotal = 0.00017f;

	// kg * m^2
		public const float m_Azimuth_Inertia1 = m_Azimuth_InertiaTotal / 2f;
									// kg *
m^2
		public const float m_Azimuth_Inertia2 = m_Azimuth_Inertia1 *
(m_Azimuth_GearRatio * m_Azimuth_GearRatio);// kg * m^2

		public const float m_Azimuth_ViscousFriction2 = 0.0026f;

					// N * m / (rad / s)
		public const float m_Azimuth_ViscousFriction1 =
m_Azimuth_ViscousFriction2 * ((m_Azimuth_GearRatio - 1) / m_Azimuth_GearRatio);  //
N * m / (rad / s)

		public const float m_Azimuth_CoulombFriction1 = 0.1f;      // N * m
		public const float m_Azimuth_CoulombFriction2 = 36f;       // N * m

		public const float m_Azimuth_MaximumTorque = 30f;
					// N * m
		public const float m_Azimuth_MaximumSpeed = (1200f / 60f) *
MathHelper.TwoPi;      // rad / s
		public const float m_Azimuth_TorsionConstant = 10f;
					// N * m / rad
		private float m_Azimuth_Backlash = 0.000070f;
						// rad

		// Position
		private float m_Latitude;
		private float m_Longitude;
	}
}
```

B.4 Source code – Environment.cs

```
using System;
using Microsoft.Xna.Framework;

namespace SatelliteAntenna
{
	public class Environment
	{
		public Environment()
		{
			m_Random = new Random();
			m_WindDirection = new Vector3();
		}

		public void Think()
		{
			if (Simulation.Time >= m_WindThink)
			{
				// Select time for next wind property change
				m_WindThink = Simulation.Time + 10 + (m_WindStrength /
6);
```

```
                              // Select new pseudo-random direction
                              float DirChangeCoeff = (float)m_Random.NextDouble() *
2 - 1;
                              m_WindYawTarget += ((float)Math.PI + (m_WindStrength *
0.05f)) * DirChangeCoeff * DirChangeCoeff * DirChangeCoeff * 0.25f;
                              m_WindPitchTarget = DirChangeCoeff * 0.03f;
                              m_WindSpeedTarget = m_WindStrength;
                       }

                       // Change wind direction gradually
                       m_WindYaw += ((m_WindYawTarget - m_WindYaw) * Simulation.Step
* 0.3f);
                       m_WindPitch += ((m_WindPitchTarget - m_WindPitch) *
Simulation.Step * 0.3f);
                       m_WindSpeed += ((m_WindSpeedTarget - m_WindSpeed) *
Simulation.Step * 0.4f);

                       float CosPitch = (float)Math.Cos(m_WindPitch);
                       m_WindDirection.X = (float)Math.Cos(m_WindYaw) * CosPitch;
                       m_WindDirection.Y = (float)Math.Sin(m_WindYaw) * CosPitch;
                       m_WindDirection.Z = (float)Math.Sin(m_WindPitch);
               }

               public const float Radius = 6371000;
               public const float Period = 86164.091f; // 23 hours 56 minutes 4.091
seconds

               private Random m_Random;

               private float m_WindStrength;
               private float m_WindThink;

               private float m_WindYaw;
               private float m_WindPitch;
               private float m_WindSpeed;

               private Vector3 m_WindDirection;

               private float m_WindYawTarget;
               private float m_WindPitchTarget;
               private float m_WindSpeedTarget;
       }
}
```

B.5 Source code – Satellite.cs

```
using System;
using Microsoft.Xna.Framework;

namespace SatelliteAntenna
{
       public class Satellite
       {
               public Satellite()
               {
                       m_Inclination = Angle.ToRadian(15);
                       m_Altitude = 200000;
                       m_Radius = Environment.Radius + m_Altitude;
                       m_Period = 1800;

                       m_Longitude = Angle.ToRadian(90);
               }

               public void Think()
               {
                       // Pre-multiplied rotation matrices used to calculate
                       // the position of satellite w.r.t. the antenna
```

```
                        float Phase = m_PhaseShift + 2 * (float)Math.PI *
Simulation.Time / m_Period;
                        float PhaseEarth = -0.01f + 2 * (float)Math.PI *
Simulation.Time / Environment.Period;

                        float SinLatitudeAnt = (float)Math.Sin(-
Program.Antenna.Latitude);
                        float CosLatitudeAnt = (float)Math.Cos(-
Program.Antenna.Latitude);
                        float SinPhaseEarth = (float)Math.Sin(PhaseEarth +
Program.Antenna.Longitude - m_Longitude);
                        float CosPhaseEarth = (float)Math.Cos(PhaseEarth +
Program.Antenna.Longitude - m_Longitude);
                        float SinInclinationSat = (float)Math.Sin(m_Inclination);
                        float CosInclinationSat = (float)Math.Cos(m_Inclination);
                        float SinPhaseSat = (float)Math.Sin(Phase);
                        float CosPhaseSat = (float)Math.Cos(Phase);

                        m_Position.X = (CosLatitudeAnt * CosPhaseEarth *
CosInclinationSat + SinLatitudeAnt * SinInclinationSat) * CosPhaseSat +
CosLatitudeAnt * SinPhaseEarth * SinPhaseSat;
                        m_Position.Y = -SinPhaseEarth * CosInclinationSat *
CosPhaseSat + CosPhaseEarth * SinPhaseSat;
                        m_Position.Z = (SinLatitudeAnt * CosPhaseEarth *
CosInclinationSat - CosLatitudeAnt * SinInclinationSat) * CosPhaseSat +
SinLatitudeAnt * SinPhaseEarth * SinPhaseSat;

                        m_Position *= m_Radius;

                        Vector3 V = m_Position - new Vector3(Environment.Radius, 0,
0);

                        if ((V.Y == 0) && (V.Z == 0))
                        {
                               m_Azimuth = 0;
                               m_Elevation = (float)Math.PI / 2;
                        }
                        else
                        {
                               Vector3 VProj = new Vector3(0, V.Y, V.Z);

                               m_Azimuth = (float)Math.Atan2(VProj.Z, VProj.Y);

                               float Cosine = Vector3.Dot(V, VProj) / (V.Length() *
VProj.Length());
                               if (Cosine > 1)
                                      Cosine = 1;

                               float Acos = (float)Math.Acos(Cosine);

                               m_Elevation = Acos * Math.Sign(V.X);
                        }

                        m_MeasuredAzimuth = m_Azimuth;
                        m_MeasuredElevation = m_Elevation;
                 }


                 // Orbit
                 // http://www.lns.cornell.edu/~seb/celestia/orbital-parameters.html
                 // Apocenter distance = Pericenter distance
                 // Time of pericenter passage = irrelevant
                 private float m_Altitude;
                 private float m_Inclination; // Angle of inclination
                 private float m_Longitude; // Longitude of ascending node
                 private float m_Period; // Period of orbital motion in seconds

                 private Vector3 m_Position;
```

```
		private float m_Radius; // Radius of orbit in meters

		private float m_PhaseShift;

		// Calculated
		private float m_Azimuth;
		private float m_Elevation;

		private float m_MeasuredAzimuth;
		private float m_MeasuredElevation;

		#endregion
	}
}
```

B.6 Source code – Simulation.cs

```
using System;
using System.IO;
using System.IO.Ports;
using System.Windows.Forms;
using System.Text;

namespace SatelliteAntenna
{
	public static class Simulation
	{
		#region Constructors

		static Simulation()
		{
			m_Time = 0;
			m_Step = 0.0001f;
			m_SamplingSteps = 1;
		}

		#endregion

		#region Methods

		public static void Start()
		{
			if (m_State != RunState.Stopped)
				return;

			if (Program.MainForm.cmbSettings_SerialComm_Port.SelectedItem
== null)
			{
				MessageBox.Show("Please select a COM port for the
serial communication port.", "Error", MessageBoxButtons.OK, MessageBoxIcon.Error);
				return;
			}

			m_Time = 0;
			m_TimeIndex = 0;
			m_LogIndex = 0;
			m_ScopeRefresh = 0;
			m_SatelliteRisen = false;

			// Calculate the rising and setting times of the satellite.
			// This ensures that a simulation run will always last
			// for one full flight, regardless of orbit parameters.
			float Shift;

			Shift = 0;
			Program.Satellite.PhaseShift = Shift;
			Program.Satellite.Think();
```

```
                        bool Was = (Program.Satellite.Elevation > -0.017);
                        int i;
                        for (i = 1; i < 10010; i++)
                        {
                                Shift = (float)Math.PI * i / 5000;
                                Program.Satellite.PhaseShift = Shift;
                                Program.Satellite.Think();

                                if ((Program.Satellite.Elevation > -0.017) && !Was)
                                        break;

                                Was = (Program.Satellite.Elevation > -0.017);
                        }

                        if (i == 10010)
                        {

                                if (!Was)
                                {
                                        MessageBox.Show("Satellite will never be
visible. Change orbit parameters.", "Error", MessageBoxButtons.OK,
MessageBoxIcon.Error);
                                        return;
                                }
                                else
                                {
                                        MessageBox.Show("Satellite will always be
visible. Simulation will not be automatically completed.", "Warning",
MessageBoxButtons.OK, MessageBoxIcon.Warning);
                                }
                        }

                        m_TimeIndex =
(int)((float)Program.MainForm.nudMain_Time_Start.Value / m_Step);
                        m_Time = m_TimeIndex * m_Step;
                        Program.Satellite.Think();

                        try
                        {
                                m_Writer = new
StreamWriter(Program.MainForm.txtSettings_DataFile_Name.Text, true);
                                m_Writer.WriteLine("*** Satellite Tracking - Cadmus -
v" + Program.Version.ToString() + " ***");
                                m_Writer.WriteLine("*** Logging started at " +
DateTime.Now.ToString());

                                string LogLine = m_Time.ToString("Time");

                                // Create header line based on which variables
                                // will be recorded

                                // ...
                                // ...
                                // ...

                                m_Writer.WriteLine(LogLine);
                                m_Writer.Flush();
                        }
                        catch (Exception ex)
                        {
                                MessageBox.Show(ex.Message, "Error",
MessageBoxButtons.OK, MessageBoxIcon.Error);
                                return;
                        }

                        try
                        {
                                //Program.SerialPort.ReadBufferSize = 16384;
```

```
                              Program.SerialPort.PortName =
Program.MainForm.cmbSettings_SerialComm_Port.Text;
                              Program.SerialPort.DataBits = 8;
                              Program.SerialPort.StopBits = StopBits.One;
                              Program.SerialPort.BaudRate =
Convert.ToInt32(Program.MainForm.cmbSettings_SerialComm_BaudRate.Text);
                              Program.SerialPort.Parity =
(Parity)Enum.Parse(typeof(Parity),
Program.MainForm.cmbSettings_SerialComm_Parity.Text);
                              Program.SerialPort.Open();
                      }
                      catch (Exception ex)
                      {
                              MessageBox.Show(ex.Message, "Error",
MessageBoxButtons.OK, MessageBoxIcon.Error);
                              return;
                      }

                      // Disable GUI controls for the duration of simulation
                      // ...

                      // Write initial data to registers
                      // ...

                      m_State = RunState.Running;
              }

              public static void Stop()
              {
                      if (m_State == RunState.Stopped)
                              return;

                      // Re-enable GUI controls
                      // ...

                      m_Writer.WriteLine("*** Logging finished at " +
DateTime.Now.ToString());
                      m_Writer.WriteLine();
                      m_Writer.WriteLine();
                      m_Writer.Flush();
                      m_Writer.Close();

                      if (Program.SerialPort.IsOpen)
                              Program.SerialPort.Close();

                      m_State = RunState.Stopped;
              }

              public static void AdvanceTime()
              {
                      m_TimeIndex++;
                      m_Time = m_TimeIndex * m_Step;

                      // Update scopes
                      // ...

                              // Redraw scopes every second (a slow operation)
                              if (m_Time >= m_ScopeRefresh)
                              {
                                      // ...

                                      m_ScopeRefresh = m_Time + 1;
                              }

                              m_LogIndex++;

                              if (!m_SatelliteRisen)
                                      if (Program.Satellite.Elevation >= 0.0175f)
                                              m_SatelliteRisen = true;
```

```
                        if (Program.MainForm.cbMain_Record.Checked)
                            WriteLogLine();
                    }
                }

                private static RunState m_State;
                private static float m_Time;
                private static int m_TimeIndex;
                private static float m_Step;
                private static int m_LogIndex;
                private static float m_ScopeRefresh;

                private static float m_SamplingTime;
                private static int m_SamplingSteps;
                private static float m_LogInterval;
                private static int m_LogSteps;

                private static StreamWriter m_Writer;
                private static string m_LogDelimiter;

                private static bool m_SatelliteRisen;

                #endregion
        }

        public enum RunState
        {
                Stopped,
                Running,
        }
}
```

B.7 Source code – StateSolver.cs

```
using System;

namespace SatelliteAntenna
{
        public static class StateSolver
        {
                public static float[] RungeKutta4(StateEquationDelegate
StateEquation, float[] X, float[] U, float Time, float Step)
                {
                        float[] Result = new float[X.Length];
                        float HalfStep = Step / 2;
                        float[] k1 = new float[X.Length];
                        float[] k2 = new float[X.Length];
                        float[] k3 = new float[X.Length];
                        float[] k4 = new float[X.Length];
                        float[] ModX = new float[X.Length];

                        for (int i = 0; i < X.Length; i++)
                                ModX[i] = X[i];

                        k1 = StateEquation(ModX, U, Time);

                        for (int i = 0; i < X.Length; i++)
                        {
                                k1[i] *= Step;
                                ModX[i] = X[i] + k1[i] / 2;
                        }

                        k2 = StateEquation(ModX, U, Time + HalfStep);

                        for (int i = 0; i < X.Length; i++)
                        {
                                k2[i] *= Step;
                                ModX[i] = X[i] + k2[i] / 2;
                        }

                        k3 = StateEquation(ModX, U, Time + HalfStep);

                        for (int i = 0; i < X.Length; i++)
                        {
                                k3[i] *= Step;
                                ModX[i] = X[i] + k3[i];
                        }

                        k4 = StateEquation(ModX, U, Time + Step);

                        for (int i = 0; i < X.Length; i++)
                        {
                                k4[i] *= Step;
                                Result[i] = X[i] + ((k1[i] + (2 * k2[i]) + (2 * k3[i])
+ k4[i]) / 6);
                        }

                        return Result;
                }

                public delegate float[] StateEquationDelegate(float[] X, float[] U,
float Time);
        }
}
```

B.8 Source code – Protoc16.h

```
void SA_ReadRegister(int8 id, int16 *data)
{
        putc(97 + id);
        *data = getc();
        *data <<= 8;
        *data += getc();
}

void SA_WriteRegister(int8 id, int16 *data)
{
        putc(65 + id);
        putc((int8)(*data >> 8));
        putc((int8)(*data));
}
```

B.9 Definition and pseudocode – PWM_Receive.v

```
module PWM_Receive (
        input clk,
        input PWM_clk,
        input PWM_data,
        output reg[9:0] PWM_recv_data,
        output PWM_OK);

// Pseudocode
On each clock cycle
        If sync signal is high then
                Reset width counter
        If PWM signal is high and width counter is below maximum value then
                Increment width counter
        If PWM signal falls or width counter reaches maximum value then
                Latch width counter value
                Raise data received signal
```

B.10 Definition and pseudocode – Encoder.vhd

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
use ieee.numeric_std.all;

entity Encoder2 is
        port (
                clock: in std_logic;
                input: in std_logic_vector(31 downto 0);
                trigger: in std_logic;
                chan_A: out std_logic;
                chan_B: out std_logic
        );
end Encoder2;

-- Pseudocode
On each clock cycle
        If trigger signal is high then
                Latch number of edges to be sent
                Latch rotation direction
                Latch delay counter value
        If number of edges to be sent is nonzero then
                Increment delay counter
                If delay counter value is reached then
                        If rotation direction is clockwise then
                                Increment edge counter
                        Else
                                Decrement edge counter
                        Reset delay counter
Always
        Assign to Channel A: (edge counter bit 0) xor (edge counter bit 1
        Assign to Channel B: edge counter bit 1
```

B.11 Source code – main.cpp

```
#include <fcntl.h>
#include <math.h>
#include <memory.h>
#include <unistd.h>
#include <signal.h>
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>

#include "CadmusDevice.h"
#include "CadmusManager.h"
#include "EncoderCommander.h"
#include "Solver_RK4.h"
#include "DisturbanceSource.h"

#include "simdecl.h"

int k = 0;
float step = 0.001f;
float t;
float x[SYSTEM_NUMSTATES] = INITIAL_STATES;
float u[SYSTEM_NUMINPUTS];
float x_new[SYSTEM_NUMSTATES];
```

```
char tx[40] = { 0, };
char rx[40] = { 0, };

CadmusDevice* g_Device;
CadmusManager* g_Manager;
EncoderCommander* g_Encoder;
Solver_RK4* g_Solver;
DisturbanceSource *g_Disturbance_x;
DisturbanceSource *g_Disturbance_y;
DisturbanceSource *g_Disturbance_z;

extern void stateEquation(float* x, float* u, float time, float* dx);

void simStep()
{
        k++;
        t = step * k;

        g_Solver->Step(x, u, t, x_new);

        for (int i = 0; i < SYSTEM_NUMSTATES; i++)
                x[i] = x_new[i];
}

void *signal_thread(void *p)
{
        while(k < 40500)
        {
                g_Device->WaitInterrupt();

                // Receive inputs
                g_Device->Receive((uint8_t*)rx, 6);

                u[0] = ((float)(*(unsigned short*)&rx[0]) - 512.0f) * 0.0625f;
                u[1] = ((float)(*(unsigned short*)&rx[2]) - 512.0f) * 0.0625f;
                u[2] = ((float)(*(unsigned short*)&rx[4]) - 512.0f) * 0.0625f;
                u[3] = g_Disturbance_x->GetValue(k);
                u[4] = g_Disturbance_y->GetValue(k);
                u[5] = g_Disturbance_z->GetValue(k);

                // Step
                simStep();

                // Calculate encoder information
                g_Encoder->SetPosition(0, x[3]);
                g_Encoder->SetPosition(1, x[7]);
                g_Encoder->SetPosition(2, x[11]);

                *((unsigned int*)&tx[8]) = g_Encoder->GetCommand(0);
                *((unsigned int*)&tx[4]) = g_Encoder->GetCommand(1);
                *((unsigned int*)&tx[0]) = g_Encoder->GetCommand(2);

                g_Encoder->Advance();

                // Write outputs
                g_Device->Send((uint8_t*)tx, 12);
        }

        pthread_exit(NULL);
}

int main(int argc, char** argv)
```

```
{
        g_Device = new CadmusDevice();
        g_Device->Open();

        g_Solver = new Solver_RK4(stateEquation, SYSTEM_NUMSTATES, SYSTEM_NUMINPUTS,
step);

        g_Manager = new CadmusManager();
        g_Manager->SetServer("10.0.0.50", 5202);
        g_Manager->Prepare((1 + SYSTEM_NUMSTATES + SYSTEM_NUMINPUTS) *
sizeof(float), 1500);
        g_Manager->Connect();
        g_Manager->Start();

        g_Encoder = new EncoderCommander();
        g_Encoder->SetSamplingTime(0.001);
        g_Encoder->SetEdgesPerRev(0, 40000);
        g_Encoder->SetEdgesPerRev(1, 40000);
        g_Encoder->SetEdgesPerRev(2, 40000);
        g_Encoder->SetPosition(0, x[3]);
        g_Encoder->SetPosition(1, x[7]);
        g_Encoder->SetPosition(2, x[11]);
        g_Encoder->Advance();

        g_Disturbance_x = new DisturbanceSource();
        g_Disturbance_x->LoadDatFile("/cadmus/heavy_full_150_Fx.dat", 30000);
        g_Disturbance_y = new DisturbanceSource();
        g_Disturbance_y->LoadDatFile("/cadmus/heavy_full_150_Fy.dat", 30000);
        g_Disturbance_z = new DisturbanceSource();
        g_Disturbance_z->LoadDatFile("/cadmus/heavy_full_150_Fz.dat", 30000);

        pthread_t sigthr;
        pthread_create(&sigthr, NULL, signal_thread, NULL);

        timespec timeOut, remains;
        int nsret;

        while (k < 40500)
        {
                timeOut.tv_sec = 0;
                timeOut.tv_nsec = 200000000;

                do
                {
                        nsret = nanosleep(&timeOut, &remains);
                        timeOut.tv_sec = remains.tv_sec;
                        timeOut.tv_nsec = remains.tv_nsec;
                } while (nsret == -1);
        }

        g_Device->Close();
        g_Manager->Stop();
        g_Manager->Disconnect();

        return 0;
}
```

B.12 Source codes for NGW100 platform – CadmusDevice.h

```
#include <stdint.h>
#include <pthread.h>
#include <semaphore.h>
#include <linux/types.h>
#include <linux/spi/spidev.h>

class CadmusDevice
{
private:
        int fileDesc;
        bool isOpen;

        sem_t intSem;

        static const char *spiDevice;
        uint8_t spiMode;
        uint8_t spiBits;
        uint32_t spiSpeed;
        uint16_t spiDelay;
        int spiFileDesc;
        spi_ioc_transfer spiXfer;

        void SetSignalHandler(void (*sigHandler)(int));

        static CadmusDevice *mainDevice;

        static void SignalHandler(int signo);

public:
        CadmusDevice();
        ~CadmusDevice();

        bool Open();
        void Close();

        const char* GetDeviceName();
        const int GetMajorNumber();
        const int GetFileDesc();
        bool IsOpen();

        bool Send(uint8_t *data, int length);
        bool Receive(uint8_t *data, int length);

        void WaitInterrupt();
};
```

B.13 Source code – CadmusDevice.cpp

```
#include "CadmusDevice.h"
#include "cadmusio.h"

#include <errno.h>
#include <fcntl.h>
#include <memory.h>
#include <unistd.h>
#include <signal.h>
#include <stdlib.h>
#include <sys/ioctl.h>
#include <linux/spi/spidev.h>
```

```
const char* CadmusDevice::spiDevice = "/dev/spidev1.1";
CadmusDevice *CadmusDevice::mainDevice = NULL;

void CadmusDevice::SignalHandler(int signo)
{
        if (signo == SIGIO)
                sem_post(&mainDevice->intSem);
}

CadmusDevice::CadmusDevice()
{
        fileDesc = -1;
        isOpen = false;

        spiMode = 0;
        spiBits = 8;
        spiSpeed = 10000000;
        spiDelay = 0;

        spiXfer = *(new spi_ioc_transfer());
}

CadmusDevice::~CadmusDevice()
{
}

void CadmusDevice::SetSignalHandler(void (*sigHandler)(int))
{
        if (sigHandler != NULL)
        {
                struct sigaction action;
                memset(&action, 0, sizeof(action));
                action.sa_handler = sigHandler;
                action.sa_flags = 0;

                sigaction(SIGIO, &action, NULL);
                fcntl(fileDesc, F_SETOWN, getpid());
                fcntl(fileDesc, F_SETFL, fcntl(fileDesc, F_GETFL) | FASYNC);
        }
        else
        {
                fcntl(fileDesc, F_SETFL, fcntl(fileDesc, F_GETFL) &~ FASYNC);
                fcntl(fileDesc, F_SETOWN, -1);
        }
}

bool CadmusDevice::Open()
{
        fileDesc = open(CADMUSIO_DEVICE_NAME, 0);

        if (fileDesc < 0)
                return false;

        // *******
        // * SPI *
        // *******
        int retval;

        spiFileDesc = open(spiDevice, O_RDWR);

        // Mode
        retval = ioctl(spiFileDesc, SPI_IOC_WR_MODE, &spiMode);
        if (retval == -1)
                return false;

        retval = ioctl(spiFileDesc, SPI_IOC_RD_MODE, &spiMode);
        if (retval == -1)
                return false;

        // Bits per word
        retval = ioctl(spiFileDesc, SPI_IOC_WR_BITS_PER_WORD, &spiBits);
        if (retval == -1)
                return false;

        retval = ioctl(spiFileDesc, SPI_IOC_RD_BITS_PER_WORD, &spiBits);
        if (retval == -1)
                return false;

        // Max speed Hz
        retval = ioctl(spiFileDesc, SPI_IOC_WR_MAX_SPEED_HZ, &spiSpeed);
        if (retval == -1)
                return false;

        retval = ioctl(spiFileDesc, SPI_IOC_RD_MAX_SPEED_HZ, &spiSpeed);
        if (retval == -1)
                return false;

        sem_init(&intSem, 0, 0);

        mainDevice = this;
        SetSignalHandler(CadmusDevice::SignalHandler);

        isOpen = true;
        return true;
}

void CadmusDevice::Close()
{
        isOpen = false;

        SetSignalHandler(NULL);
        sem_post(&intSem);
        mainDevice = NULL;

        close(fileDesc);
        close(spiFileDesc);
}

const char* CadmusDevice::GetDeviceName()
{
        return CADMUSIO_DEVICE_NAME;
}

const int CadmusDevice::GetMajorNumber()
{
        return CADMUSIO_MAJOR_NUM;
}

const int CadmusDevice::GetFileDesc()
{
        return fileDesc;
}
```

```
bool CadmusDevice::IsOpen()
{
        return isOpen;
}

bool CadmusDevice::Send(uint8_t *data, int length)
{
        ioctl(fileDesc, CADMUSIO_SET_MODE, 2);

        spiXfer.tx_buf = (unsigned long)data;
        spiXfer.rx_buf = (unsigned long)NULL;
        spiXfer.len = length;
        spiXfer.delay_usecs = spiDelay;
        spiXfer.speed_hz = spiSpeed;
        spiXfer.bits_per_word = spiBits;

        if(ioctl(spiFileDesc, SPI_IOC_MESSAGE(1), &spiXfer) == 1)
                return false;
        else
                return true;
}

bool CadmusDevice::Receive(uint8_t *data, int length)
{
        ioctl(fileDesc, CADMUSIO_SET_MODE, 3);

        spiXfer.tx_buf = (unsigned long)NULL;
        spiXfer.rx_buf = (unsigned long)data;
        spiXfer.len = length;
        spiXfer.delay_usecs = spiDelay;
        spiXfer.speed_hz = spiSpeed;
        spiXfer.bits_per_word = spiBits;

        if(ioctl(spiFileDesc, SPI_IOC_MESSAGE(1), &spiXfer) == 1)
                return false;
        else
                return true;
}

void CadmusDevice::WaitInterrupt()
{
        sem_wait(&intSem);
}
```

B.14 Source code – CadmusManager.h

```
#include <netinet/in.h>
#include <pthread.h>
#include <semaphore.h>

struct StateSenderInfo
{
        char* buffer;
        int objectCount;
        size_t objectSize;
        char* fillPtr;
        char* sendPtr;
        char* bufferEnd;
        bool sendEnabled;

        int socketfd;
        sem_t* semaphore;
};

class CadmusManager
{
private:
        int socketfd;
        struct sockaddr_in serverAddress;
        bool connected;

        StateSenderInfo stateBuffer;
        pthread_t stateSendThread;
        sem_t stateSemaphore;

public:
        CadmusManager();
        ~CadmusManager();

        void SetServer(const char* address, int port);
        bool Connect();
        bool Disconnect();
        bool Prepare(size_t objectSize, int count);
        bool Start();
        bool Stop();
        bool SendStatePacket(const void* stateObject);
};

void* StateSendLoop(void* arg);

enum PacketType
{
        PACK_PING = 0x00,
        PACK_STATE = 0xff
};
```

B.15 Source code – CadmusManager.cpp

```
#include "CadmusManager.h"
#include "Log.h"

#include <arpa/inet.h>
#include <errno.h>
#include <malloc.h>
#include <memory.h>
#include <sys/socket.h>
#include <sys/types.h>
#include <unistd.h>

CadmusManager::CadmusManager()
{
        socketfd = -1;
        connected = false;
}

CadmusManager::~CadmusManager()
{

}

void CadmusManager::SetServer(const char* address, int port)
{
        memset(&serverAddress, 0, sizeof(serverAddress));
        serverAddress.sin_family = AF_INET;
        serverAddress.sin_addr.s_addr = inet_addr(address);
        serverAddress.sin_port = htons(port);
}

bool CadmusManager::Connect()
{
        if (!connected)
        {
                socketfd = socket(AF_INET, SOCK_STREAM, 0);

                if (socketfd < 0)
                        return false;

                int result = connect(socketfd, (struct sockaddr*)&serverAddress,
sizeof(serverAddress));

                if (result < 0)
                        return false;

                stateBuffer.socketfd = socketfd;
                connected = true;
                return true;
        }

        return false;
}

bool CadmusManager::Disconnect()
{
        if (connected)
        {
                int result = shutdown(socketfd, SHUT_RDWR);

                if (result < 0)
                        return false;

                result = close(socketfd);

                if (result < 0)
                        return false;

                connected = false;
                return true;
        }

        return false;
}

bool CadmusManager::Prepare(size_t objectSize, int count)
{
        int result = sem_init(&stateSemaphore, 0, 1);

        if (result < 0)
                return false;

        stateBuffer.semaphore = &stateSemaphore;
        stateBuffer.objectSize = objectSize;
        stateBuffer.objectCount = count;
        stateBuffer.buffer = (char*)malloc(objectSize * count);
        if (stateBuffer.buffer == NULL)
                return false;

        stateBuffer.bufferEnd = stateBuffer.buffer + (objectSize * count);
        stateBuffer.fillPtr = stateBuffer.buffer;
        stateBuffer.sendPtr = stateBuffer.buffer;

        stateBuffer.socketfd = socketfd;

        return true;
}

bool CadmusManager::Start()
{
        stateBuffer.sendEnabled = true;
        int result = pthread_create(&stateSendThread, NULL, StateSendLoop,
(void*)(&stateBuffer));

        if (result != 0)
                return false;

        return true;
}

bool CadmusManager::Stop()
{
        stateBuffer.sendEnabled = false;
        pthread_join(stateSendThread, NULL);
        return true;
}

bool CadmusManager::SendStatePacket(const void* stateObject)
{
        sem_wait(&stateSemaphore);

        memcpy( (void*)stateBuffer.fillPtr, stateObject, stateBuffer.objectSize);

        stateBuffer.fillPtr += stateBuffer.objectSize;

        if (stateBuffer.fillPtr >= stateBuffer.bufferEnd)
                stateBuffer.fillPtr = stateBuffer.buffer;

        // Fullcheck
        if (stateBuffer.fillPtr == stateBuffer.sendPtr)
        {
                Log::SetLastError(-1, "CadmusManager::Send(): stateBuffer full");
                return false;
        }

        sem_post(&stateSemaphore);

        return true;
}

void* StateSendLoop(void* arg)
{
        StateSenderInfo *sb = (StateSenderInfo*)arg;
        char* fillPtr;
        int semVal;

        while (sb->sendEnabled)
        {
```

```
			usleep(1000);
			sem_getvalue(sb->semaphore, &semVal);

			if (semVal > 0)
			{
				fillPtr = sb->fillPtr;

				if (fillPtr > sb->sendPtr)
				{
					send(sb->socketfd, sb->sendPtr, fillPtr - sb->sendPtr,
0);

					sb->sendPtr = fillPtr;
				}
				else if (fillPtr < sb->sendPtr)
				{
					send(sb->socketfd, sb->sendPtr, sb->bufferEnd - sb-
>sendPtr, 0);

					send(sb->socketfd, sb->buffer, fillPtr - sb->buffer,
0);

					sb->sendPtr = fillPtr;
				}
			}
		}

		return 0;
}
```

B.16 Source code – DisturbanceSource.h

```
class DisturbanceSource
{
private:
	float* data;
	int length;
	bool repeat;

public:
	void LoadDatFile(const char* fileName, size_t count);
	void LoadDatFile(const char* fileName, long int start, size_t count);
	float GetValue(int index);
};
```

B.17 Source code – DisturbanceSource.cpp

```
#include <stdio.h>
#include <malloc.h>
#include "DisturbanceSource.h"

void DisturbanceSource::LoadDatFile(const char *fileName, size_t count)
{
	data = (float*)malloc(count * sizeof(float));

	FILE *fp = fopen(fileName, "rb");

	fread(data, sizeof(float), count, fp);

	fclose(fp);

	length = count;
	repeat = true;
}

void DisturbanceSource::LoadDatFile(const char* fileName, long int start, size_t
count)
{
        data = (float*)malloc(count * sizeof(float));

        FILE *fp = fopen(fileName, "rb");

        fseek(fp, start * sizeof(float), SEEK_SET);
        fread(data, sizeof(float), count, fp);

        fclose(fp);

        length = count;
        repeat = true;
}

float DisturbanceSource::GetValue(int index)
{
        if (repeat)
        {
                return data[index % length];
        }
        else
        {
                if (index < length)
                        return data[index];
                else
                        return data[length - 1];
        }
}
```

B.18 Source code – EncoderCommander.h

```
class EncoderCommander
{
private:
        unsigned int delayBase;
        float multiplier[5];
        int oldInt[5];
        int newInt[5];

        unsigned int command;
        unsigned int dir;
        unsigned int count;
        unsigned int delay;

public:
        void SetSamplingTime(float t);
        void SetEdgesPerRev(int i, int edges);
        void SetPosition(int i, float theta);
        int GetIntegerPosition(int i);
        unsigned int GetCommand(int i);
        void Advance();
};
```

B.19 Source code – EncoderCommander.cpp

```
#include <stdio.h>
#include <math.h>
#include "EncoderCommander.h"

void EncoderCommander::SetSamplingTime(float t)
{
	delayBase = (int)(50000000.0f * t);
}

void EncoderCommander::SetEdgesPerRev(int i, int edges)
{
	multiplier[i] = (float)edges / (2 * M_PI);
}

void EncoderCommander::SetPosition(int i, float theta)
{
	newInt[i] = (int)(theta * multiplier[i]);
}

int EncoderCommander::GetIntegerPosition(int i)
{
	return oldInt[i];
}

unsigned int EncoderCommander::GetCommand(int i)
{
	if (newInt[i] > oldInt[i])
	{
		dir = 0x00000000;
		count = newInt[i] - oldInt[i];
		delay = delayBase / count;
	}
	else if (newInt[i] < oldInt[i])
	{
		dir = 0x80000000;
		count = oldInt[i] - newInt[i];
		delay = delayBase / count;
	}
	else
	{
		dir = 0x80000000;
		count = 0;
		delay = 0xffff;
	}

	command = dir | (count << 16) | delay;

	return command;
}

void EncoderCommander::Advance()
{
	for (int i = 0; i < 5; i++)
		oldInt[i] = newInt[i];
}
```

B.20 Source code – Solver_RK4.h

```
class Solver_RK4
{
private:
	void (*eqn)(float*, float*, float, float*);
	int xLen;
	int uLen;
	float* k1;
	float* k2;
	float* k3;
	float* k4;
	float* xMod;
	float stepSize;
	float halfStep;

public:
	Solver_RK4(void (*equation)(float*, float*, float, float*), const int x_len,
const int u_Len, const float step_size);
	~Solver_RK4();

	void Step(float* x, float* u, float time, float* x_new);
};
```

B.21 Source code – Solver_RK4.cpp

```
#include "Solver_RK4.h"
#include <stdlib.h>

Solver_RK4::Solver_RK4(
		void (*equation)(float*, float*, float, float*),
		const int x_len, const int u_len, const float step_size)
{
	eqn = equation;
	xLen = x_len;
	uLen = u_len;

	k1 = (float*)malloc(x_len * sizeof(float));
	k2 = (float*)malloc(x_len * sizeof(float));
	k3 = (float*)malloc(x_len * sizeof(float));
	k4 = (float*)malloc(x_len * sizeof(float));
	xMod = (float*)malloc(x_len * sizeof(float));

	stepSize = step_size;
	halfStep = stepSize / 2.0f;
}

Solver_RK4::~Solver_RK4()
{
	free(k1);
	free(k2);
	free(k3);
	free(k4);
	free(xMod);
}

void Solver_RK4::Step(float* x, float* u, float time, float* x_new)
{
	float timePhs = time + halfStep;
	float timePs = time + stepSize;
	int i;
```

```
        for (i = 0; i < xLen; i++)
                xMod[i] = x[i];

        eqn(xMod, u, time, k1);

        for (i = 0; i < xLen; i++)
        {
                k1[i] *= stepSize;
                xMod[i] = x[i] + (k1[i] / 2.0f);
        }

        eqn(xMod, u, timePhs, k2);

        for (i = 0; i < xLen; i++)
        {
                k2[i] *= stepSize;
                xMod[i] = x[i] + (k2[i] / 2.0f);
        }

        eqn(xMod, u, timePhs, k3);

        for (i = 0; i < xLen; i++)
        {
                k3[i] *= stepSize;
                xMod[i] = x[i] + k3[i];
        }

        eqn(xMod, u, timePs, k4);

        for (i = 0; i < xLen; i++)
        {
                k4[i] *= stepSize;
                x_new[i] = x[i] + ((k1[i] + (2.0f * (k2[i] + k3[i])) + k4[i]) /
6.0f);
        }
}
```

B.22 Definition and pseudocode – NGW_Interface.vhd

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
use ieee.numeric_std.all;

entity NGW_Interface is
        port (
                clk: in std_logic;
                ---------------------
                spi_sck: in std_logic;
                spi_mosi: in std_logic;
                spi_miso: out std_logic;
                spi_ssel: in std_logic;
                ---------------------
                mode: in std_logic_vector(2 downto 0);
                ---------------------
                rsignal: in std_logic;
                tsignal: out std_logic;
                ---------------------
                data_out: out std_logic_vector(319 downto 0);
                conf_out: out std_logic_vector(319 downto 0);
```

```
               data_in: in std_logic_vector(319 downto 0);
               conf_in: out std_logic_vector(319 downto 0)
        );
end NGW_Interface;

-- Pseudocode
On each clock cycle
        If SPI data is incoming then
               If mode is "write" then
                      Bit-shift sensor emulation data into registers
               If mode is "read" then
                      Bit-shift input command data through SPI to NGW100
        If SPI data transfer is completed and mode is "write" then
               Initiate sensor emulator operation
        IF signal is received from InputSignaller then
               Read input receiver data into registers
```

B.23 Definition and pseudocode – InputSignaller.vhd

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
use ieee.numeric_std.all;

entity InputSignaller is
        port (
               clock: in std_logic;
               input0: in std_logic;
               input1: in std_logic;
               input2: in std_logic;
               collect: out std_logic;
               send: out std_logic
        );
end InputSignaller;

-- Pseudocode
On each clock cycle
        For all input modules connected
               If module reception signal is high then raise module flag
        If all module flags are raised then
               Signal manager module to collect input data
               Signal NGW100 to execute simulation
               Reset all flags
```

B.24 MATLAB script – hils_milling_distgen.m

```
% Define parameters
t = 0:0.001:30;
Fmax = 1000;
harm = 4;
rpm = 150;
phase = 10;
name = 'light';

% Derived parameters
tm = (rpm/60)*t*2*pi;
Fxy_max = Fmax;
Fz_max = Fmax / 10;
phase1 = phase * (pi/180);
phase2 = phase1 + (pi/2);

% Generate values
```

```
Fx = (0.5 * Fxy_max) + (0.4 * Fxy_max * ((sin(harm * tm) + 1) / 2)) + (0.1 *
Fxy_max * ((sin(tm + phase1) + 1) / 2));
Fy = (0.5 * Fxy_max) + (0.4 * Fxy_max * ((sin(harm * tm) + 1) / 2)) + (0.1 *
Fxy_max * ((cos(tm + phase1) + 1) / 2));
Fz = (0.5 * Fz_max)  + (0.4 * Fz_max  * ((sin(harm * tm) + 1) / 2)) + (0.1 * Fz_max
* ((sin(tm + phase2) + 1) / 2));

% Normalize max. value
Fr = (Fx.^2 + Fy.^2 + Fz.^2).^0.5;
scale = Fmax / max(Fr);
Fx = Fx .* scale;
Fy = Fy .* scale;
Fz = Fz .* scale;
Fr = (Fx.^2 + Fy.^2 + Fz.^2).^0.5;

% Save binaries
fname = sprintf('%s_Fx.dat', name);
fid = fopen(fname, 'wb');
fwrite(fid, Fx, 'float32');
fclose(fid);
fname = sprintf('%s_Fy.dat', name);
fid = fopen(fname, 'wb');
fwrite(fid, Fy, 'float32');
fclose(fid);
fname = sprintf('%s_Fz.dat', name);
fid = fopen(fname, 'wb');
fwrite(fid, Fz, 'float32');
fclose(fid);
```