CORA Cork Open Research Archive
Cartlann Taighde Oscailte Chorcaí

Title	Improved kinematic sensing for motion control applications
Authors	Boggarpu, Naveen Kumar
Publication date	2015
Original Citation	Boggarpu, N. K. 2015. Improved kinematic sensing for motion control applications. PhD Thesis, University College Cork.
Type of publication	Doctoral thesis
Rights	© 2015, Naveen Kumar Boggarpu. - http://creativecommons.org/licenses/by-nc-nd/3.0/
Download date	2026-11-07 12:37:17
Item downloaded from	https://hdl.handle.net/10468/2145



UCC
University College Cork, Ireland
Coláiste na hOllscoile Corcaigh

UNIVERSITY COLLEGE CORK

Improved Kinematic Sensing for Motion Control Applications

by

Naveen Kumar Boggarpu

A thesis submitted in partial fulfillment for the degree of Doctor of Philosophy

in the

School of Engineering

September 2015

Declaration of Authorship

I, NAVEEN KUMAR BOGGARPU, declare that this thesis titled, 'IMPROVED KINEMATIC SENSING FOR MOTION CONTROL APPLICTIONS' and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given and I have acknowledged all main sources of help. With the exception of such quotations, this thesis is entirely my own work.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed:

Date:

Abstract

New compensation methods are presented that can greatly reduce the slit errors (i.e. transition location errors) and interval errors induced due to non-idealities in optical incremental encoders (square-wave). An M/T-type, constant sample-time digital tachometer (CSDT) is selected for measuring the velocity of the sensor drives. Using this data, three encoder compensation techniques (two pseudoinverse based methods and an iterative method) are presented that improve velocity measurement accuracy. The methods do not require precise knowledge of shaft velocity. During the initial learning stage of the compensation algorithm (possibly performed in-situ), slit errors/interval errors are calculated through pseudoinverse-based solutions of simple approximate linear equations, which can provide fast solutions, or an iterative method that requires very little memory storage. Subsequent operation of the motion system utilizes adjusted slit positions for more accurate velocity calculation.

In the theoretical analysis of the compensation of encoder errors, encoder error sources such as random electrical noise and error in estimated reference velocity are considered. Initially, the proposed learning compensation techniques are validated by implementing the algorithms in MATLAB software, showing a 95% to 99% improvement in velocity measurement. However, it is also observed that the efficiency of the algorithm decreases with the higher presence of non-repetitive random noise and/or with the errors in reference velocity calculations.

The performance improvement in velocity measurement is also demonstrated experimentally using motor-drive systems, each of which includes a field-programmable gate array (FPGA) for CSDT counting/timing purposes, and a digital-signal-processor (DSP). Results from open-loop velocity measurement and closed-loop servocontrol applications, on three optical incremental square-wave encoders and two motor drives, are compiled. While implementing these algorithms experimentally on different drives (with and without a flywheel) and on encoders of different resolutions, slit error reductions of 60% to 86% are obtained (typically approximately 80%).

Extended Abstract

New compensation methods are presented that can greatly reduce the slit errors (i.e. transition location errors) and interval errors induced due to non-idealities in optical incremental encoders (square-wave) caused by manufacturing limitations in the code wheel, optical components and analog circuitry. An M/T-type, constant sample-time digital tachometer (CSDT) is selected for measuring the velocity of the sensor drives. This CSDT measurement technique, involving pulse-count and high-frequency timer measurement, effectively time-stamps the encoder transitions, in turn almost eliminating the quantization error. Using CSDT-based data, three encoder compensation techniques (two pseudoinverse based methods and an iterative method) are presented that improve velocity measurement accuracy. One of the main advantages of these methods is that they do not require precise knowledge of shaft velocity, thereby eliminating the need for high-accuracy reference equipment. Instead, this work adopts three different mathematical solutions (mean velocity calculation, interpolation and zero-phase filter techniques) to calculate the reference velocity. During the initial learning stage of the compensation algorithm (possibly performed in-situ), slit errors/interval errors are calculated through pseudoinverse-based solutions of simple approximate linear equations - which can provide fast solutions, or an iterative method that requires very little memory storage. Subsequent operation of the motion system utilizes adjusted slit positions for more accurate velocity calculation.

In the theoretical analysis of the compensation of encoder errors, encoder error sources such as random electrical noise and error in estimated reference velocity

are considered. The inability to distinguish between small repetitive physical velocity changes over a mechanical cycle and the alteration in the transition errors over the same mechanical cycle, which correlates with velocity changes, is shown to lead to a remaining error, even after compensation.

Initially, the proposed learning compensation techniques are validated by implementing the algorithms in MATLAB software. For simulation analysis, an encoder containing all the non-idealities discussed in the theoretical analysis is simulated. Using this simulated encoder, for a known velocity profile, the required raw data obtained from the CSDT method is calculated. The required reference velocity profiles are generated from CSDT data by using either a zero-phase filter, interpolation, or an averaging technique. These MATLAB implementations show a 95% to 99% improvement in velocity measurement. However, it is also observed that the efficiency of the algorithm decreases with the higher presence of non-repetitive random noise and/or with the errors in reference velocity calculations.

The performance improvement in velocity measurement is also demonstrated experimentally using motor-drive systems, each of which includes a field-programmable gate array (FPGA), for implementing the velocity measurement technique, and a digital-signal processor (DSP), for implementing the learning algorithm and control loop. Results from open-loop velocity measurement and closed-loop servo control applications on three optical incremental square-wave encoders and two motor drives are compiled. The performances of the three compensation algorithms are also compared for different reference velocity estimation techniques,

where the zero-phase filter technique is shown to provide the best possible reference velocity estimation. It is observed that a reasonably good estimate of slit/interval errors can be performed with less than 2000 samples, using two similar pseudoinverse-based methods. The experimental results obtained are consistent with those expected from a theoretically modeled encoder. While implementing these algorithms experimentally on different drives (with and without a flywheel) and on encoders of different resolutions, slit error reductions of 60% to 86% are obtained (typically approximately 80%) with a similar improvement in velocity measurement is also observed. Reasons for the variation in the error reduction between simulations and experimental results are discussed.

Acknowledgements

First of all I would like to express my deepest appreciation to my Supervisor, Senior Lecturer Richard C Kavanagh, for continually encouraging and guiding me during the research work and especially during the final stages of the dissertation. Without his guidance and persistent help, this dissertation would not have been possible.

I would like to thank my mother, Rukmani Guptha Boggarapu, and my father, Sathya Narayana Guptha Boggarapu for the love and affection, and for their care and sacrifice they made to put my career and future ahead of their comfort.I would like to thank My brother Nagendra Prasad Boggarapu and sister Padmaja Boggarapu for their support, especially at the critical stage of my career, My wife, Suchitra Mogili, who has provided me with constant help and motivation, especially during the final stages of this dissertation, and my son, Vivek, for several exciting and very happy moments.

In addition, I would like to thank the Gojanur family for their care during my stay in Ireland and for considering me a part of their family.

Fellow researchers at the Power Electronics lab and administration staff had been a great help, it helped me to settle down into a new environment and culture very easily and to be feel comfortable and 'one of them'. This period of four years was one of the most memorable periods of my life.

Contents

Declaration of Authorship i

Abstract ii

Extended Abstract iii

Acknowledgements vi

List of Figures x

List of Tables xiv

Abbreviations xv

Symbols xvii

1 Introduction 1

1.1 Background of the Project 1

1.2 Encoder Types and Common Errors 4

1.2.1 Square-wave encoders 5

1.2.2 Sinusoidal encoders 6

1.3 Description of the Error Sources of Optical (Square Wave) Incremental Encoders 7

1.3.1 Quantization error 11

1.4 Basic Speed Measuring Methodologies 12

1.4.1 Elapsed-time method 14

1.4.2 Pulse-count tachometer - **PCT** 15

1.4.2.1 Quadrature decoding 16

1.5 Advanced Velocity Measurement Techniques 17

1.5.1 M/T method 18

1.5.2 Method of accurate time-interval measurement 19

1.5.3 CSDT method 21

1.5.3.1 Low-speed CSDT 24

1.5.3.2 Oversampled constant sample-time digital tachometer (OCSDT) . . . 25
1.6 Previous and Proposed Work on Compensation of Encoder Outputs 26
1.7 Hardware Implementation . . . 30
1.7.1 Optical encoders used in experimental work . . . 31
1.7.2 Field Programmable Gate Array . . . 32
1.7.2.1 The Xilinx XC4000 FPGA . . . 33
1.7.3 dSPACE DS1104 . . . 36
1.7.4 Electrical drives used in experimental work . . . 37
1.8 Aim and Objectives . . . 39
1.8.1 Objectives . . . 39
1.9 Outline of the Thesis . . . 40

2 Description of the Learning Algorithm 42
2.1 Introduction . . . 42
2.2 Code-wheel Error - Description and Nomenclature . . . 47
2.3 Description of the Pseudoinverse-Based Learning Algorithms . . . 49
2.3.1 Description of Pseudoinverse-A method . . . 49
2.3.2 Description of Pseudoinverse-B method . . . 54
2.4 Iterative Solution of the Error Equations . . . 58
2.5 Conclusions . . . 62

3 Error Modeling and Analysis 64
3.1 Introduction . . . 64
3.2 Error Modeling with Random Noise Included . . . 65
3.2.1 Error modeling considering slit error, $\delta_{P(i)}$ . . . 67
3.3 Effect of sinusoidally distributed error on shaft velocity . . . 71
3.4 Simulation of Encoder, Tachometer and Iterative Learning . . . 74
3.4.1 Investigation of various means of calculating the reference velocity . . . 80
3.4.2 Behaviour of slit and interval errors . . . 81
3.4.3 Use of real encoder data in tachometer simulation . . . 83
3.5 Conclusions . . . 85

4 Implementation and Evaluation of Learning Algorithms 87
4.1 Introduction . . . 87
4.2 Implementation and Experimental Evaluation of Learning Techniques in Open-Loop Measurement Applications . . . 88
4.2.1 Implementation of learning technique in open-loop . . . 93
4.2.2 Performance analysis of the proposed algorithms in an open-loop implementation . . . 95
4.2.2.1 Improvements in velocity estimates . . . 96
4.2.2.2 Performance and output of the learning algorithm . 100
4.2.3 Investigation of alternative means of calculating the reference velocity . . . 104

4.2.4 Experimental evaluation of the noise and errors associated with an encoder . . . 106
4.3 Experimental Results for a Closed-Loop Servosystem . . . 110
4.3.1 Design of the closed-loop controller . . . 111
4.3.2 Comparative results to verify utility of the learning algorithm in a closed-loop system . . . 112

5 Conclusions **115**
5.1 Conclusions . . . 115

A Architecture of DS1104 dSPACE R & D Controller Board **119**

Bibliography **121**

List of Figures

1.1 Basic block diagram of a digital drive setup. 3
1.2 A typical square-wave incremental encoder code-wheel. 5
1.3 Output waveforms of a typical 3 channel incremental square-wave encoder, with Channel-A leading Channel-B. 6
1.4 Output waveforms of a typical incremental sinusoidal encoder, where Channel-A is leading Channel-B. 7
1.5 Output waveforms of two nominally quadrature Incremental Encoder channels (reproduced from [1]). 8
1.6 Time-diagram shown (a) with, and (b) without, a codewheel window covered with dust. 11
1.7 Plots showing effects of numerical rounding of signal x in (a), with corresponding error in (b), and truncation in (c), with corresponding error plot (d). 13
1.8 Elapsed-time digital tachometer timing diagram (considering positive-going transitions of one channel only). 14
1.9 Pulse count digital tachometer timing diagram. 15
1.10 Time diagram showing the generation of an ideal quadrature decoded pulse from encoder channels A and B. 17
1.11 M/T digital tachometer timing diagram. 18
1.12 Time-stamping concept used in Nutt's Digital Time Intervalometer [2]. . 20
1.13 Timing diagram of CSDT method for measurement interval $T(i)$ (assuming measurement based on positive-going transitions only of encoder channel A or B); the digital positions at samples i-1 and i (marked as s_{i-1} and s_i) are denoted by $P(i-1)$ and $P(i)$, respectively. 22
1.14 Timing diagram of low-speed CSDT method for measurement interval $T(i)$ (assuming measurement based on positive-going transitions only of encoder channel A or B); the digital positions at samples i-6 and i (marked as s_{i-6} and s_i) are denoted by $P(i-1)$ and $P(i)$, respectively. Note, $n = 6$ is assumed in this example. 24
1.15 Time diagram showing the sub-sampling during one sampling interval. . 25
1.16 Concept of time-stamping used by Merry to calculate the quantized time at a sample instant to determine actual physical position(unquantized). . 29
1.17 Timing diagram of SR method showing the time determination for calculating average shaft velocity (times displayed are based on measurement of positions 1 and 2. 30
1.18 Block diagram of a typical FPGA/DSP-based servosystem controller. . . 31

1.19 Block diagram of a basic FPGA (Extracted from Xilinx data book). . . 34
1.20 Simplified logic diagram of Configurable Logic Block (CLB) (slightly modified extract from Xilinx data book). 35

2.1 A graph showing the calculated residual in a system against the number of iterations (reproduced from [3]). 44
2.2 Non-ideal code wheel with eight counts per revolution showing ideal (integer) and actual transition positions, where position 0 (which corresponds to the encoder's zero marker) provides the reference. 48
2.3 Comparison of CSDT velocity and the reference velocity generated using a zero-phase filter. (Example from a MATLAB-generated, simulated CSDT/encoder) 52

3.1 Actual and calculated slit errors for nominal encoder transition positions of 100 to 150, with the learning algorithm incorporated into the simulation program. 77
3.2 Simulation outputs comparing actual shaft velocity with CSDT velocities, before and after compensation, (for $T_s = 1$ ms). 77
3.3 Performance of iterative algorithm for varied amounts of random noise, using simulated encoder data 78
3.4 Performance of iterative algorithm with different proportions of added low-frequency noise, using real encoder data 79
3.5 Slit error profiles generated from a MATLAB-simulated incremental encoder 82
3.6 Interval error profile generated by a MATLAB-simulated incremental encoder 82
3.7 Probability density function of slit and interval errors generated from a MATLAB simulated incremental encoder 84
3.8 Performance of iterative algorithm with different proportions of added high-frequency noise 85
3.9 Comparing the effect of position independent error, $\delta^n(i)$, on the figure of merit for a simulated encoder of 360 ppr (where magniture of position-independent error is of the order Case 3 < Case 1 < Case 2). 86

4.1 Block diagram of a typical FPGA/DSP-based servosystem controller. . . 88
4.2 Block diagram of a FPGA/DSP-based CSDT implementation (reproduced for convenience). 89
4.3 Block diagram showing the overall setup of the implementation. 91
4.4 Block diagram of a typical FPGA/DSP-based servosystem controller (reproduced for convenience). 92
4.5 Estimated shaft velocity for Omron encoder (360 ppr) connected to high-inertia load, with MATLAB-based implementation of iterative learning algorithm. 95
4.6 Estimated shaft velocity for OVW encoder (360 ppr), using DSP-based implementation of iterative learning algorithm. 96

4.7 Velocities derived from flywheel-mounted quadrature decoded HP encoder signals; (MATLAB-based learning implementation). (Note: Two counts per sample time equates to 0.5 counts per sample interval without quadrature decoding). 97
4.8 Estimated error reduction for high inertia system (using Omron and OVW encoders) at different velocities when using iterative and Pseudoinverse B algorithms, with look-up tables calculated at the same speeds at which the compensation is performed, with the exception of the trace marked with '◇', for which the look-up table is calculated using a ramp-down traverse from 460 rpm to 310 rpm. 98
4.9 Estimated error reduction for low inertia system (using HP encoder) at different velocities, with look-up tables calculated at same velocities at which the compensation is performed, with the exception of the trace marked '*', where the look-up table is calculated during a ramp-up traverse from 2040 rpm to 2310 rpm. 99
4.10 Estimated error reduction for high inertia system (using Omron and OVW encoders) at different velocities when using iterative (Ite), Pseudoinverse B (Psu B) and Merry's [4, 5] algorithms (Mer), with look-up tables calculated at the same speeds at which the compensation is performed 100
4.11 Calculation of estimated error reduction (using Omron encoder with 360 ppr) at 170 rpm, for various sample lengths, N_s. 101
4.12 Calculation of estimated error reduction (using OVW encoder with 360 ppr) at 170 rpm, for various sample lengths, N_s. 101
4.13 Calculation of estimated error reduction at 1700 rpm for HP encoder (200 ppr), without flywheel, as N_s varies. 102
4.14 Slit errors of OVW encoder's transition positions 100 to 150 for positive-going edge of channel B in clockwise, and negative-going edge in anti-clockwise, directions, at 170 rpm. 103
4.15 Convergence of interval error estimates using the iterative learning algorithm, to steady-state values, for a few randomly selected interval errors at 170 rpm, using an Omron encoder, with 360 ppr. 103
4.16 Estimated error reduction, using Pseudoinverse B method with Omron encoder (360 ppr), for variously calculated reference velocities. 105
4.17 Root-mean square of high-frequency error in the compensated shaft velocity at three different band-widths obtained by implementing the Pseudoinverse-A learning algorithm on the Omron encoder. 106
4.18 Probability density function of interval and slit errors of the Omron 360 ppr incremental encoder, calculated using the iterative learning algorithm. 108
4.19 Probability density function of interval error for Omron incremental encoder, and comparison with some mathematical models (with resolution of 360 ppr, calculated using iterative learning algorithm). 109
4.20 Block diagram of a typical FPGA/DSP-based servosystem controller (reproduced for convenience from Section 1.7). 110

4.21 Torque ripple, showing effect of switching velocity feedback source from uncompensated CSDT, to compensated CSDT and, finally, to PCT, when using 200 ppr HP encoder rotating at 2000 rpm, (the d.c. motor torque is 0.0294 Nm approximately). 112

A.1 DS1104 R & D Controller Board Block Diagram. 120

List of Tables

1.1 List of selected encoders that are utilized for experimental analysis . 32
1.2 Electrical drives selected to build the servo system 39

2.1 Sequence of operations performed to implement the Pseudoinverse-A learning algorithm 55
2.2 Sequence of operations performed to implement Pseudoinverse-B learning algorithm 58
2.3 Sequence of operations performed to implement Iterative learning algorithm, assuming positive shaft velocity 63

4.1 Incremental encoder selected for experimental testing 93
4.2 Comparative study of observed experimental characteristics and judiciously selected mathematical models performance 109
4.3 Servosystem Parameters used for Experimental Implementation . . 111
4.4 Normalized RMS Value of High-Frequency Torque Ripple in Servosystem 113

5.1 Normalized RMS Value of High-Frequency Torque Ripple in Servosystem 117

Abbreviations

ADC	**A**nalog to **D**igital **C**onverter
ASIC	**A**pplication **S**pecfic **I**ntegrated **C**ircuits
cps	**c**ounts **p**er **s**econd
CLK	**H**igh speed **C**lock
CLB	**C**onfigurable **L**ogic **B**lock
CSDT	**C**onstant **S**ample-time **D**igital **T**achometer
DAC	**D**igital to **A**nalog **C**onverter
DSP	**D**igital **S**ignal **P**rocessor
EPROM	**E**rasable **P**rogrammable **R**ead **O**nly **M**emory
EPA	**Embedded P**olygons **A**lgorithm
ETT	**Elapsed-T**ime **T**achometer
FG	**F**unction **G**enerator
FPGA	**F**ield **P**rogrammable **G**ate **A**rray
IOB	**I**nput **O**utput **B**lock
ISA	**I**ndustry **S**tandrad **A**rchitecture
LED	**L**ight **E**mitting **D**iode
LUT	**L**ook **U**p **T**able

OSC	**O**scillator
OSE	**O**verlapping **S**ubgraph **E**stimator
p.d.f	**p**robability **d**ensity **f**unction
ppr	**p**ulses **p**er **r**evolution
pps	**p**ulses **p**er **s**ample
PCT	**P**ulse **C**ount **T**achometer
PI	**P**roportional **I**ntegral
PLD	**P**rogrammable **L**ogic **D**evice
PPC	**P**ower**PC**
PWM	**P**ulse **W**idth **M**odulation
rpm	**r**evolutions **p**er **m**inute
Rop	**O**ptical **R**adius
RAM	**R**andom **A**ccess **M**emory
RDBK	**R**ead**B**ack
SSI	**S**mall **S**cale **I**ntegration
VHDL	**VHSIC H**ardware **D**escription **L**anguage
ZM	**Z**ero **M**arker

Symbols

a_s	Constant representing a simple sinusoidally distributed error
$\mathbf{A}$	Sparse coefficient matrix relating $\mathbf{e}$ and Δ_A
$\mathbf{A}^+$	Pseudoinverse of matrix $\mathbf{A}$
$\mathbf{B}$	Sparse coefficient matrix relating $\mathbf{e}$ and Δ_B
$\mathbf{B}^+$	Pseudoinverse of matrix $\mathbf{B}$
c	Cycle - length of time or electrical angle between two consecutive raising edges on either Channel-A or Channel-B
$C_a(i)$	Auxiliary count at i^{th} sample instant
C_h	Number of high-speed clock counts
d	Deviation of mis-aligned detector of an encoder from the nominal location
$e(i)$	Estimated differential slit-error at i^{th} sample
$\mathbf{e}$	Estimated differential slit-error vector
f	Frequency of FPGA clock (Hz)
f_a	Frequency of digital timer (Hz)
f_c	Frequency of high-frequency clock
J	Inertia of permanent magnet synchronous motor
k	Constant to produce OCLK signal

K	Constant to scale the shaft velocity to a physical unit
K_{enc}	Encoder gain
K_t	Torque constant
K_a	Amplifier gain
K_{dac}	DAC gain
K_{vp}	PI controller's proportional gain
K_{vi}	PI controller's integral gain
K_{pl}	Plant gain
L	Resolution of a square-wave incremental encoder
m	Figure of merit, as a percentage, for the error reduction in shaft velocity using learning algorithms
$\acute{m}$	Apparent figure of merit, is the percentage of error reduction in shaft velocity using learning algorithms
m_1	Number of encoder counts generated during detection time T_d
m_2	Number of high-frequency clock pulses generated during detection time T_d
$M(i)$	Digital count change over a sample i
M^o	Average pulse count over a sampling interval
n	Number of time samples since the previous position change
N_s	Total number of samples
p_1/p_2	Pulse width - electrical angle over which a channel output is high during a cycle, nominally 180 electrical degrees
P_i	Ideal transition positions of an encoder

$\bar{P}_i$	Actual transition positions of an encoder
$P(i)$	Integer (digital) position at i^{th} sample
$\bar{P(i)}$	Actual encoder position at i^{th} sample
q	Represents signal quantization
r	Residual value
s_1/s_2	State width - Electrical angle between transitions on Channel A or B and the adjacent Channel B or A, norminally 90 electrical degrees
S_i	i^{th} sample instant
t_i	Time instant i
$T_a(i)$	Auxiliary time for first pulse in a sample interval used in time-stamping or CSDT-based technique, at i^{th} sample
$T_b(i)$	Auxiliary time for most recent pulse in a sample interval used in time-stamping or CSDT-based technique
T_c	Nominal measurement time in M/T method
T_d	Detection time in M/T method
T_{di}	Time between the present i^{th} and immediately previous encoder transition position
T_a^o	Average auxiliary time over a sampling interval
T_s	Sample time at which the encoder information, position and auxilary time, are calculated
$V(i)$	Average shaft velocity using CSDT method over i^{th} sample interval (pps)
$V_{act}(i)$	Average actual shaft velocity, V_{act}, at i^{th} sample interval (pps)
$V_c(i)$	Calculated average shaft velocity after compensation using CSDT

	method (pps)
$\bar{V}_{err}(i)$	Average actual velocity error before compensation over i^{th} sample interval (pps)
$\bar{V}_{err-c}(i)$	Average actual velocity error after compensation over i^{th} sample interval (pps)
$\hat{V}_{err}(i)$	Apparent velocity error in shaft velocity before compensation over i^{th} sample interval (pps)
$\hat{V}_{err-c}(i)$	Apparent velocity error in shaft velocity after compensation over i^{th} sample interval (pps)
$V_I(i)$	Apparent shaft velocity using CSDT method over i^{th} sample interval (pps)
$V_{M/T}$	Average shaft velocity calculated using M/T method (rpm)
$V_{mean}(i)$	Mean shaft velocity, at i^{th} sample interval (pps)
$V_{noise}(i)$	Average shaft velocity at i^{th} sample, including slit/interval error disturbance due to random noise (pps)
v_{OC}	Oversampled CSDT velocity
$V_r(i)$	Average reference shaft velocity, V_r, at i^{th} sample interval (pps)
x_i	Encoder position at sample i
x_q	Quantized output of signal q
y	Index variable used iterative method
α	Variable gain term in iterative method
Δ_A	Transpose matrix of line position errors
Δ_B	Transpose matrix of interval width errors

Δc	Cycle error
δ_k	Slit error of k^{th} transition location
$\acute{\delta}_k$	Transition error estimate at k^{th} transition location
$\delta_{k,j}$	Interval error between k^{th} and j^{th} transition locations
$\delta^-_{k,j}$	Uncompensated component of interval error between k^{th} and j^{th} transition locations
δ^n	Random noise which is independent of encoder line position
$\delta^n_{P(i)}$	Position-dependent random noise at encoder transition position $P(i)$
$\delta^{lf}_{P(i)}$	Low-frequency error induced at position $P(i)$ due to code-wheel eccentricity
$\delta^r_{P(i)}$	Position error at transition position $P(i)$, due to the estimation of reference velocity
Δp	Pulse width error
Δs	State error
$\Delta\phi$	Phase error
ϵ	Quantized error occured due to quantization of signal q
ϕ	Number of electrical degrees between the middle of the high state outputs of two channels
ϕ	Phase - number of electrical degrees between the middle of the high state outputs of two channel (A and B), nominally 90 electrical degrees

Dedicated to my mother and my father

Chapter 1

Introduction

1.1 Background of the Project

With the emergence of a highly competitive market, the industrial sector has evolved from manual to automated industrial processes. This change has lead to the development of various electrical drives, operating in open- and closed-loop systems. For the accurate measurement of position/velocity and stable servosystems, different feedback sensors (low-power transducers) such as analog tachometers, digital tachometers, encoders and resolvers have been used in these drives. Initially, resolvers and analog tachometers were widely-used sensor devices in servosystems. With the advent of digital control techniques, digital tachometers, which are more efficient and require less maintenance, have emerged as a viable alternative for providing feedback signals. Though low-power, d.c. permanent magnet tachogenerators are still widely used for shaft speed measurement, additional analog-to-digital converters are needed when these are included in digital

control systems. In many modern servo systems, the concept of velocity feedback has been eliminated in order to decrease the parts count of control hardware, and hence, production cost. However, for very high-bandwidth applications such as very-low-speed servo controllers and NC machines, very fast and accurate shaft speed measurement is required for stable control of the system.

The optical encoder or other sensor (which feedbacks the position/velocity of the shaft in a digital control system) is a major component of the digital tachometer. However, the accuracy of an optical encoder is influenced by the optical, mechanical and electrical parts of the sensor, including their corresponding manufacturing limitations [1]. Non-idealities in these parts lead to different types of errors, which can be broadly classified into low-frequency and high-frequency errors. However, low-frequency position error components are inconsequential for the accuracy of a velocity servosystem, due to the negligible associated measured velocity error caused by the differentiation of the position error. Having identified the high-frequency non-idealities in the encoder as a principle error source, the research reported in this thesis involves the provision of a software solution that is capable of analyzing the raw data from the encoder and predicting and compensating for the high-frequency encoder error, without any additional measurement equipment.

A basic schematic of a closed-loop digital servo system is shown in Fig. 1.1, where an encoder that is connected to a motor provides position/velocity information. This information will be used as feedback for the control that is implemented in a digital processor. Based of the error between a reference signal and the feedback from an encoder, new torque command signals typically provide current-loop

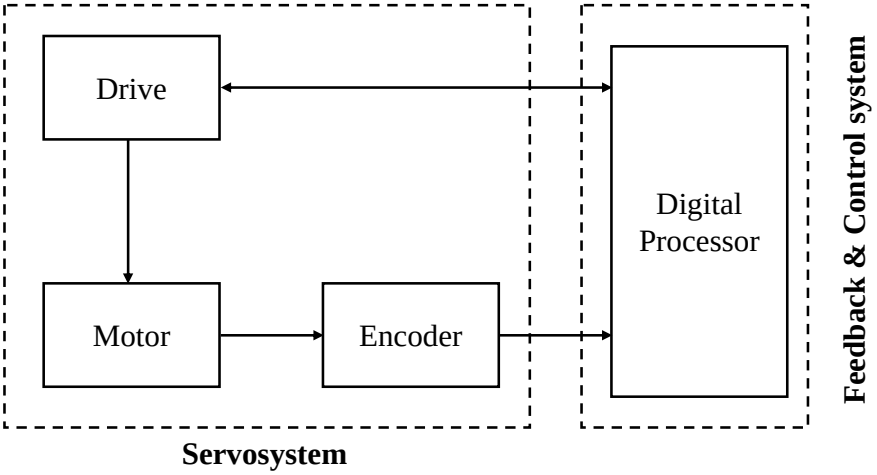


FIGURE 1.1: Basic block diagram of a digital drive setup.

inputs for the drive which, in turn, supplies the pulse-width-modulated (PWM) voltages applied to the motor under control.

In order to implement this closed-loop servosystem, a selection of various hardware and software components and a speed measuring algorithm, are required. As a matter of relevance, different types of typical encoder are discussed first in Section 1.2, and some suitable encoders that fulfil the requirements are selected for testing. Various possible speed measuring techniques are discussed in Sections 1.4 and 1.5. A technique which allows the measurement of the shaft speed with high precision, and which is easy to implement, with minimal equipment requirement, is selected. Additionally, a detailed description of the hardware utilized for the project, such as encoders, electrical drives, motors, and a digital control system, is presented in Section 1.7.

1.2 Encoder Types and Common Errors

The rotary, or shaft, encoder is one of the most commonly used types of devices to measure the speed of servo systems, with optical encoders being predominantly used. These optical encoders can be classified as incremental and absolute encoders [6–8]. An absolute encoder, which is usually optically-based, (though capacitive and magnetic encoders are occasionally used) has a code-wheel that generates a binary code [8–10]. Mechanical encoders are no longer used due to their requirement for debouncing circuitry. The binary code generated can be either a standard binary code or a Gray code. The code-wheel etched with a binary code has many concentric circular tracks depending on resolution; (resolution of a L track encoder will be 2^L). As the cost of the absolute encoder is very high due to the complexity involved in manufacturing, the absolute encoder is usually not an attractive proposition for high-volume usage. The proposed learning algorithms described in this thesis for incremental-encoder-based systems could equally be applied to an absolute encoder-based system, with very minor modifications.

Because the incremental encoder has two output channels at (normally) 90 electrical to each other, they are also known as quadrature encoders [11]. Depending on their outputs, optical incremental encoders are classified into two types: square-wave encoders and sinusoidal encoders.

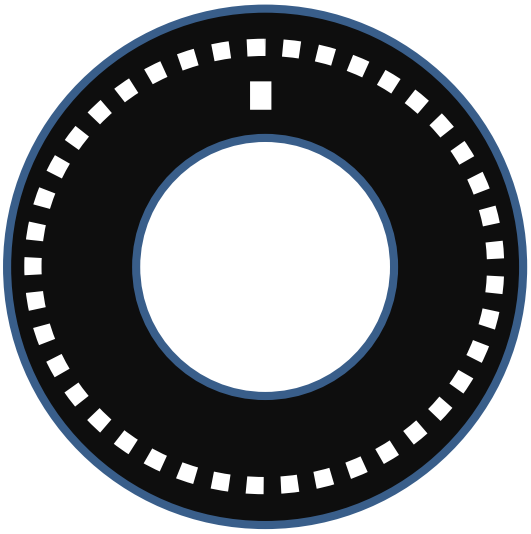


FIGURE 1.2: A typical square-wave incremental encoder code-wheel.

1.2.1 Square-wave encoders

Square-wave incremental encoders, whose output waveforms are nominally square in shape, are very commonly used, due to the inherent simplicity and noise immunity of digital signals. These square-wave incremental encoders can be either two or three channel devices, where two of these channels/output waveforms, typically labelled as Channel-A and Channel-B, are square-wave outputs which appear in quadrature. Typical output waveforms of a square-wave encoder are shown in Fig. 1.3. If Channel-A leads Channel-B in one direction, Channel-B will lead Channel-A in the opposite direction. This quadrature arrangement can be used to detect the direction to the shaft rotation and also serves to increase the resolution of the encoder by implementing the quadrature-decoding technique, as described in detail in Section 1.4.2.1. The third channel is a zero-marker (ZM), which typically resets a pulse position counter once per revolution.

Though the basic signals generated by the photodetectors of an optical incremental

square-wave encoder are pseudo-sinusoidal in shape, these sinusoidal signals are converted to square-wave signals via of Schmitt triggers, or a similar squaring technique. Clearly, the resolution, L, of a square-wave encoder is based on the

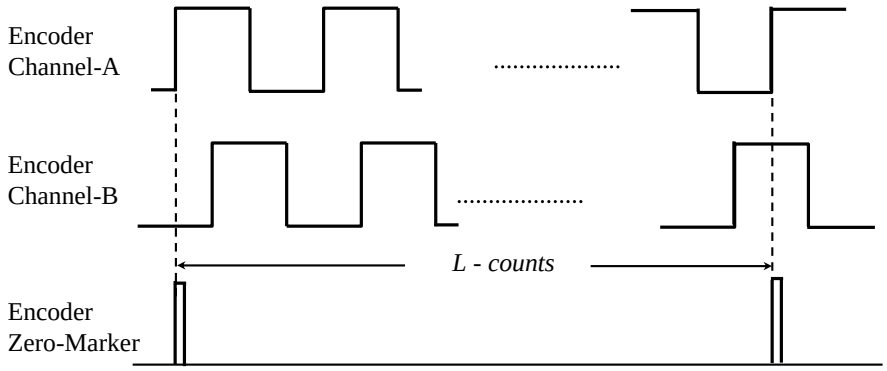


FIGURE 1.3: Output waveforms of a typical 3 channel incremental square-wave encoder, with Channel-A leading Channel-B.

number of tracks/notches on the code-wheel.

1.2.2 Sinusoidal encoders

Sinusoidal encoders are another type of incremental encoder, where the output waveforms are nominally sinusoidal in shape (though this shape is sometimes closer to triangular for a low resolution encoder) [12]. Though most of the sinusoidal encoder setup is similar to that of a square-wave encoder, the major difference between them is the absence of squaring circuitry in the former encoder. Sinusoidal encoders have two sinusoidal outputs in quadrature with each other, as shown in Fig. 1.4. As most digital tachometers based on square encoders are fundamentally limited at low velocity by the low frequency of positional information provided by the digital encoder, a sinusoidal encoder which can provide ultra-high-resolution position information, and thereby improve the accuracy and transient response of

the sensor, is widely used. The raw electrical signals emitting from the photo-detectors which convert the light transmitted through the code wheel (usually quasi sinusoidal waves) provide the basis for sinusoidal encoders.

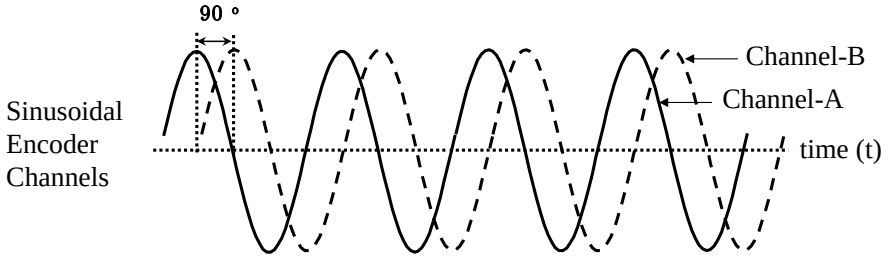


FIGURE 1.4: Output waveforms of a typical incremental sinusoidal encoder, where Channel-A is leading Channel-B.

Given that the optical, incremental square-wave encoder is simple, economical, easily used and can be adapted easily in digital systems, it was decided to use this type of encoder for testing. As the compensation of the high-frequency repetitive errors is the main objective of this thesis, it is essential to analyse the various sources of these errors. A detailed description of the different errors associated with the various parts of an incremental square-wave encoder is presented in the next section.

1.3 Description of the Error Sources of Optical (Square Wave) Incremental Encoders

While optical incremental encoders can be an inexpensive option to measure shaft velocity and position, their capability of providing accurate and reliable measurement prompted their widespread usage. Optical incremental encoders are widely

used in the fields of medical equipment, industrial robotics, computer peripherals, etc. Mechanical, electrical, and optical non-idealities determine the accuracy of an encoder. A detailed study by Yien [1] classified the incremental encoder errors as being due to pulse width error, cycle error, state width error, and phase error. These are typically derivatives of the parameters listed below.

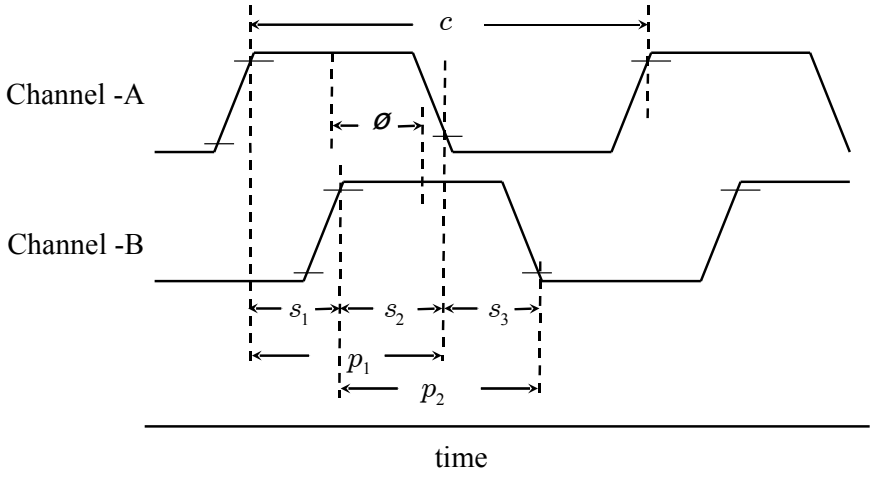


FIGURE 1.5: Output waveforms of two nominally quadrature Incremental Encoder channels (reproduced from [1]).

Encoder parameters:

- Cycle, c: Length of time or electrical angle between two consecutive rising edges of either Channel A or Channel B, nominally 360 electrical degrees.
- Counts Per Revolution, n: The number of electrical cycles per revolution.
- Pulse width, p_1/p_2: Electrical angle over which a channel output is high during a cycle, nominally 180 electrical degrees.
- State width, $s_1/s_2/s_3$: Electrical angle between transitions on Channel A or B and the adjacent Channel B or A, nominally 90 electrical degrees.

- Phase, ϕ: Number of electrical degrees between the middle of the high state outputs of two channels (A and B), nominally 90 electrical degrees.

- Optical Radius (Rop): The distance between the centre of the code-wheel and optical centre of the encoder.

The errors in some of these parameters are defined as follows:
Cycle error, Δc, is the number of electrical degrees variation from one complete (nominal) cycle, .i.e. 360 electrical degrees. Pulse width error, Δp, is the difference between the actual and ideal high state during a cycle (in electrical degrees). Similarly, the error in the state width, s, is termed the state width error, Δs and the error in phase is phase error, $\Delta\phi$. The major causes of errors in optical incremental encoders are due to eccentricity, axial play, variation in light level, mounting misalignment, code-wheel point defects, and mechanical linkage errors

Some errors deserve particular attention:

1. Code-wheel Point Defects: The code-wheel is the most critical part in an encoder, as any imperfection in a code-wheel can affect encoder performance significantly. In general, code-wheel point defects can be caused by manufacturing imperfections in window/bar edges, or the presence of dust on the code-wheel, as shown in Fig. 1.6. By using more than one LED per channel, the output can be averaged out, thereby reducing the error due to defects in the window/bar edges.

2. Variation in Light Level: The variation in the light level due to degradation of an LED source or other components, opto-electronic temperature variation, or light transmission path variation, can affect the duty cycle of the channel outputs. With the help of push-pull or differential circuitry, this effect can be minimized [13].

3. Mounting mis-alignment of photo-detectors and code-wheel (in the radial direction) will result in an output phase error. The amount of phase error can be calculated as (from [14])

$$\Delta\phi = \frac{d \cdot s \cdot n \cdot 360}{Rop^2 \cdot 2\pi} \tag{1.1}$$

where, s is the distance between two detectors (one per channel), and d is the deviation of a mis-aligned detector from the nominal location. Hence, the phase error can be reduced by having smaller distances between the detectors.

4. Shaft radial play and eccentricity cause low-frequency distortion, with a fundamental frequency corresponding to the mechanical speed of the shaft. This can be represented as an integral non-linearity. This error leads to phase or duty cycle error, and is caused when there is a mis-alignment between the axes of the code-wheel and stationary electronics, or where the axial play of the code-wheel is perpendicular to the optical path between the LEDs and photo-detectors. In addition to manufacturing defects, mis-alignment

can also occur due to stresses on the encoder shaft caused by imperfect mechanical linkage with the driving inertia.

5. Due to unequal rise and fall times of digital signals and the dependency of the digitization circuitry on the incident light intensity, the non-ideality of the electronic circuitry can cause significant errors at very high rotational speeds.

Most of these errors can be minimized, as explained in detail in [1]. However, errors due to manufacturing limitations, or to dust or dirt, cannot be removed completely. To compensate for many of the non-idealities described in this section, new learning algorithms are presented in this work.

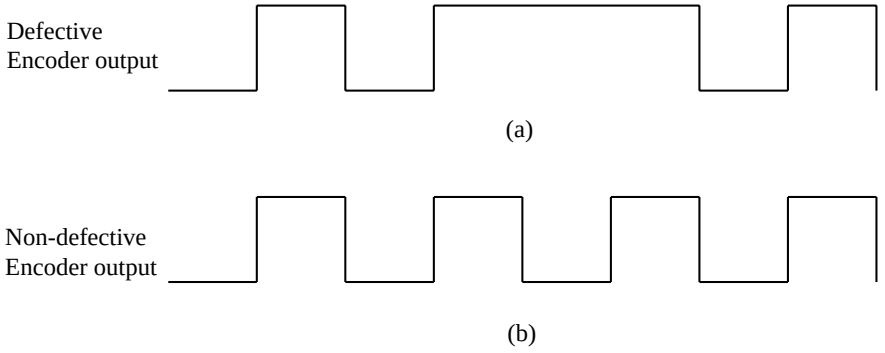


FIGURE 1.6: Time-diagram shown (a) with, and (b) without, a codewheel window covered with dust.

1.3.1 Quantization error

In additional to the various encoder errors discussed earlier, it is also necessary to consider the effects of quantization error on the accuracy of the velocity measurement. Quantization error, induced by the quantization process that arises

during analog-to-digital conversation (ADC), is defined as the difference between the analog input value and the quantized digital value. This error is a result of round-off and truncation processes in an ADC. Despite its simplicity in description and construction, the non-linear nature of the error makes it hard to analyze. Quantization noise or error is one of the major sources of errors that can restrict the resolution of data acquisition and control systems [15, 16]. The effect of quantization in communication systems, control applications, digital filters, and computational algorithms has been extensively analyzed [17–19]. The knowledge of the rate of change of digital position is very important, especially when a digital differentiator is used to estimate a velocity from sampled digital position values. The quantization function and its associated errors can be represented in a number of equivalent ways, two of which are illustrated in Fig. 1.7. In the case of Fig. 1.7(a) $x_q = q\left\lfloor \frac{x}{q} + \frac{1}{2} \right\rfloor$, where the error ϵ is defined as $x - x_q$. For Fig. 1.7(c), $x_q = q\left\lfloor \frac{x}{q} \right\rfloor$. The corresponding errors are shown in Fig. 1.7(b) and Fig. 1.7 (d). It should be noted that the error measures of the velocity estimates which result from the difference of two quantized position estimates are identical for the different choices of quantizer. It is essential to consider a velocity measurement technique that can minimize the effects of the position quantization inherent in the encoder.

1.4 Basic Speed Measuring Methodologies

Research has been going on in the field of digital tachometry for more then five decades, and several speed measuring techniques have been proposed. However, it

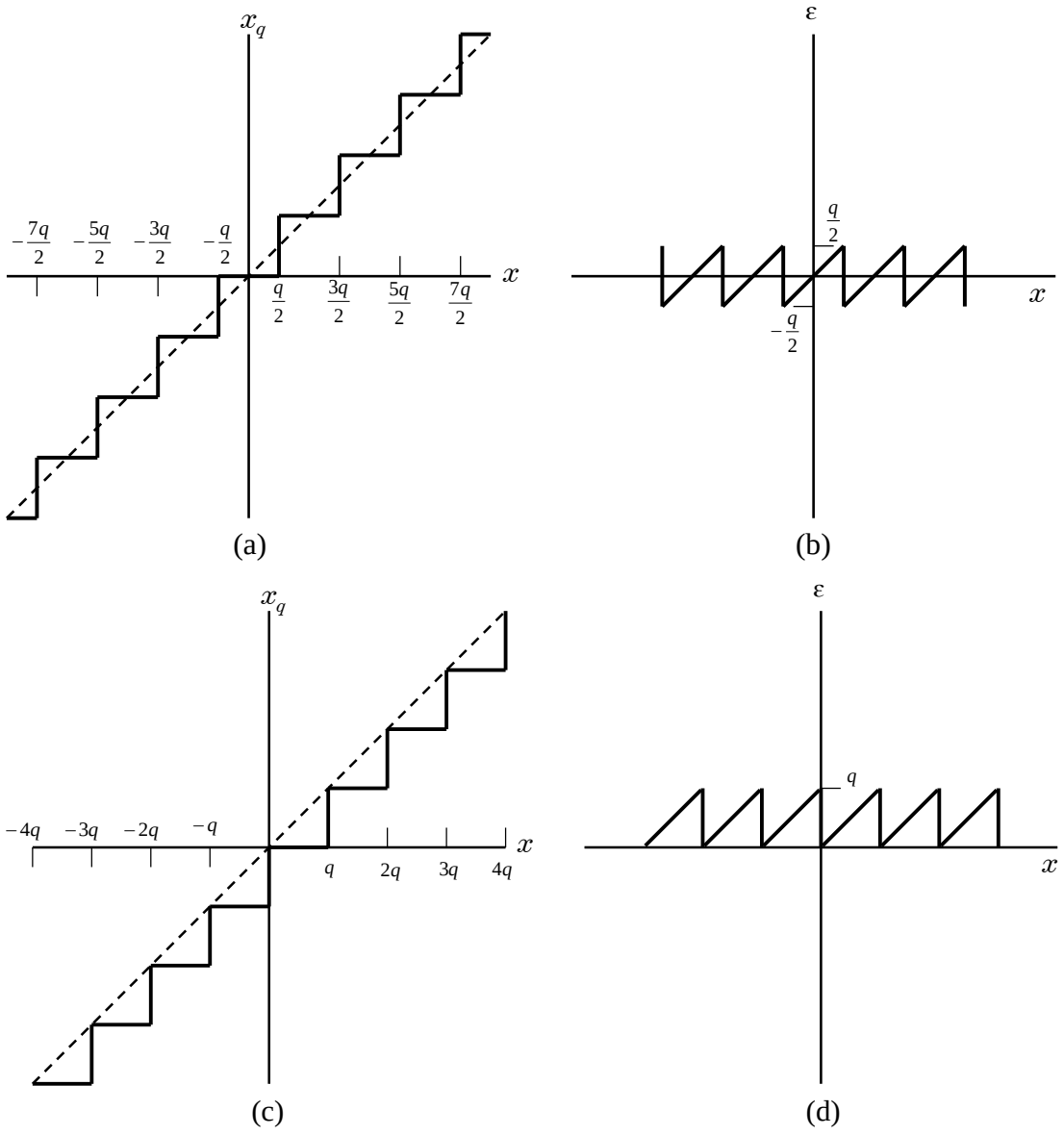


FIGURE 1.7: Plots showing effects of numerical rounding of signal x in (a), with corresponding error in (b), and truncation in (c), with corresponding error plot (d).

can be said that these methods are evolved from two basic methods: the elapsed time method and the pulse counting method [20].

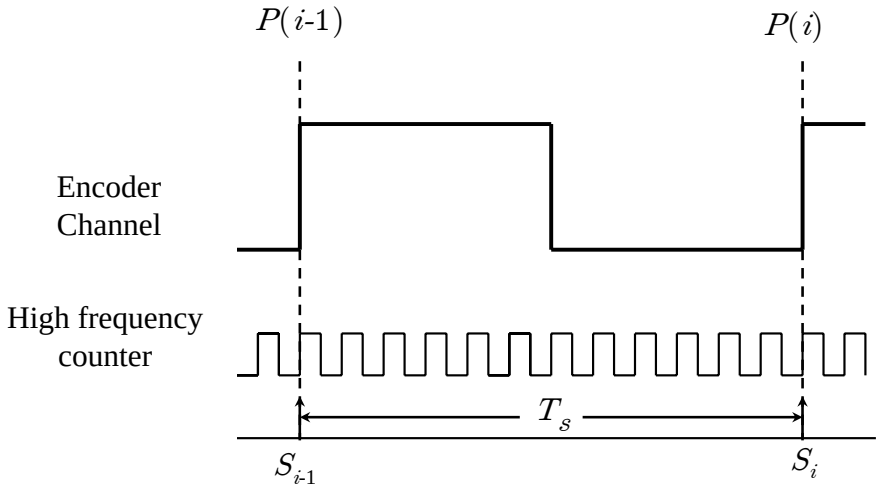


FIGURE 1.8: Elapsed-time digital tachometer timing diagram (considering positive-going transitions of one channel only).

1.4.1 Elapsed-time method

The elapsed-time method typically involves a digital time measurement over a single digital position change. This is based on an inverse function calculation, i.e. it produces a count that is inversely proportional to the shaft speed. As shown in Fig.1.8, a high-frequency clock is used to measure the time between two successive changes in digital position. The high-frequency clock starts at a given rising (or falling) edge and stops at the next rising (or falling) edge. The average shaft velocity, in r.p.m., using the elapsed-time method is

$$V_{rpm} \approx 60 \frac{f_a}{C_h L} \tag{1.2}$$

where, C_h is the number of high-speed clock counts (frequency of f_a Hz). As the average velocity between the consecutive edges is calculated, the transient behaviour of the system is easily evaluated. However, this also makes the method highly sensitive to the transition noise (errors in edge locations) inherent in the

encoder output. The resolution of the elapsed-time tachometer is determined by the frequency (f_a) of the high-speed counter and the encoder resolution. The accuracy at very high speeds may not be good, i.e. if few (or no) counts occur per measurement, and hence the influence of quantization error will be significant. The problem is minimized by increasing f_a.

1.4.2 Pulse-count tachometer - PCT

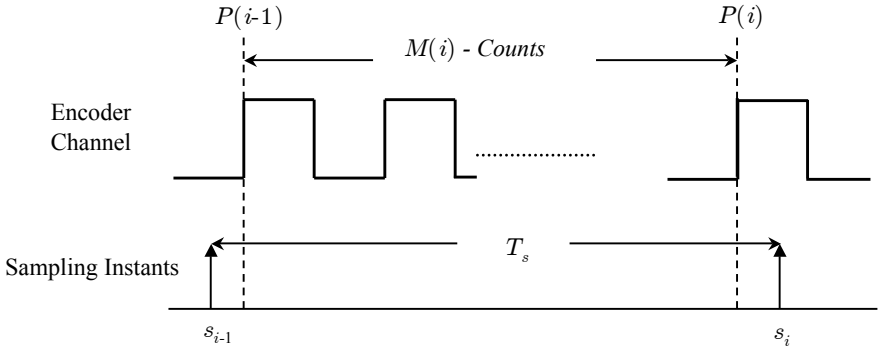


FIGURE 1.9: Pulse count digital tachometer timing diagram.

Due to its less complex and easily implemented nature, the pulse-count method is the most commonly used speed measurement technique. In this method, a fixed sample time, T_s, is used to measure the shaft speed by counting the number of counts/edges, $M(i)$, that occur over a sample interval.

$$V \approx \frac{M(i)}{T_s} \tag{1.3}$$

where V is the average shaft velocity, in counts per sample interval, T_s, (i.e. the units are encoder pulses per sample interval). Similarly, the average velocity in

r.p.m. can be expressed as

$$V_{rpm} \approx 60 \cdot \frac{M(i)}{T_s \cdot L} \ , \tag{1.4}$$

where L is the encoder resolution. This method is suitable for medium and high speed measurements. Though this method is widely used for its simplicity, quantization restricts its use at very-low speeds, and resolution can be unsatisfactory even at high-speed for demanding applications. The resolution issue at low velocities can be partly addressed by implementing quadrature decoding, as explained below. Additionally the problem can also be addressed, at the expense of poorer transition response, by using a longer sampling time, T_s.

The pulse-count method is similar to standard digital differentiation of quantized signals and is well documented in [19]. It is also worth noting that it is the most common method used in servo systems to calculate a digital velocity estimate. This involves the subtraction of successive digital velocity estimates, which gives identical results to the PCT.

1.4.2.1 Quadrature decoding

The process of increasing the resolution of a square-wave encoder by a factor of four, through consideration of both high-to-low (falling edge) and low-to-high (raising edge) transitions of both encoder channels A and B, is known as quadrature decoding. Quadrature decoding is achieved by means of suitable digital circuitry. The output signal after multiplexing will be as shown in Fig. 1.10. The shaft velocity (using the PCT method) of the servo system when using quadrature decoding

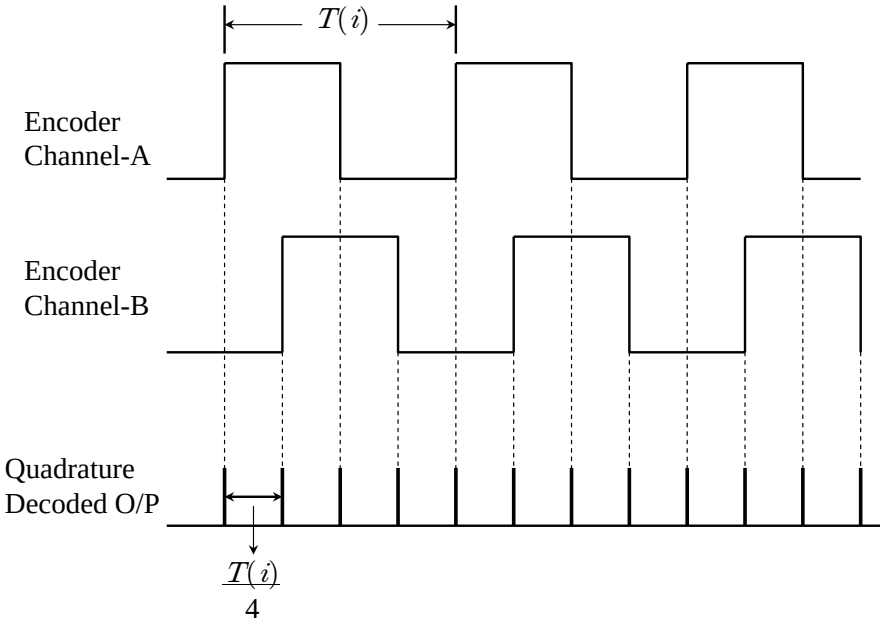


FIGURE 1.10: Time diagram showing the generation of an ideal quadrature decoded pulse from encoder channels A and B.

can be measured as

$$V(i) = \frac{K \cdot M(i)}{T(i)} = \frac{P(i) - P(i-1)}{4T_s} \tag{1.5}$$

where, $M(i)$ is number of quadrature pulses generated during the sample interval T_s. K is, a conversion constant, equal to $1/4$, and the average velocity, $V(i)$, is measured in units of encoder cycles per sample interval.

1.5 Advanced Velocity Measurement Techniques

Based on the pulse-count tachometer method and elapsed-time method, various improved velocity measuring techniques have been derived over time. Some of the more significant of these are presented in this section. A suitable velocity

measurement method will be selected for implementation, after analysis of these methods.

1.5.1 M/T method

In order to provide high accuracy and fast measurement over a wide speed range, Ohmae et.al. proposed the M/T method [21]. This method is a fusion of the two basic speed measuring techniques: (the 'M' corresponds to the pulse count method and 'T' stands for the elapsed time). This M/T method overcomes the problems in the M-method (resolution and accuracy are not so high at low-speeds) and the T-method (poor accuracy and resolution at high-speeds) and can realize high resolution and accuracy with improved bandwidth.

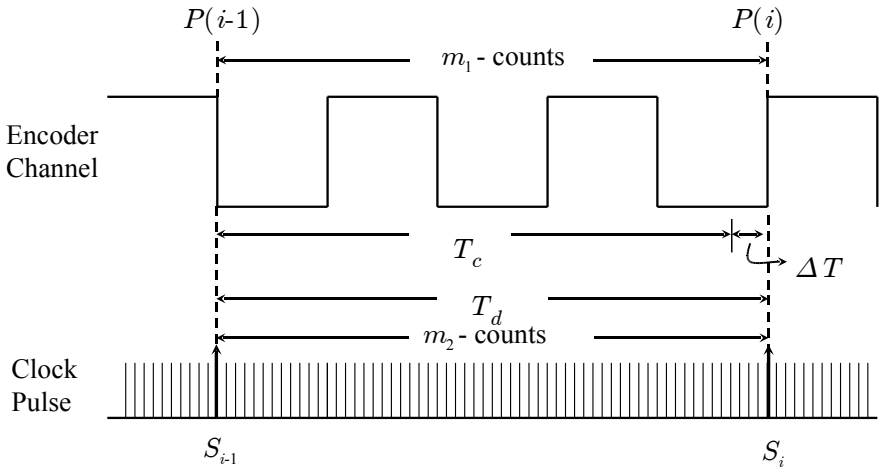


FIGURE 1.11: M/T digital tachometer timing diagram.

In Fig. 1.11, the detection time, $T_d(s)$, is determined by synchronizing the first encoder output pulse edge after the nominal sample time $T_c(s)$. It is assumed that m_1 pulses are generated during time interval $T_d(s)$ from an encoder with

a resolution of L pulses per revolution and m_2 is equivalent to the number of high-frequency counts generated, when the detection time, T_d, is digitized using a high-frequency clock of frequency f_c, (Hertz). Then, the average shaft velocity, $V_{M/T}$, (rpm) is given by,

$$V_{M/T} = 60 \cdot \frac{f_c \cdot m_1}{L \cdot m_2}. \tag{1.6}$$

1.5.2 Method of accurate time-interval measurement

The concept of interval measurement using time-stamping is common in many velocity measurement techniques [2, 4, 5, 22, 23], though the word 'time-stamping' is often not used. For precise time measurement, in [2], Nutt described an interval measurement technique that uses delay lines to measure the time between the rising edges of an oscillator and falling and raising edges of the start and end pulses. The start/stop pulse is routed through the delay line, the phase delay of which is known, and compared with an oscillator's phase to obtain the phase difference. The delay time that just exceeds the phase difference is used as the correct phase difference. By using a number of such delay lines in the system, the resolution of the measurement can be increased. This is shown in Fig. 1.12, where time-stamping is implemented to calculate auxiliary times $T_a(i)$ and $T_b(i)$ at the i^{th} sample, and the event duration time $T(i)$ can be calculated as

$$T(i) = T_s + T_a(i) - T_b(i) \tag{1.7}$$

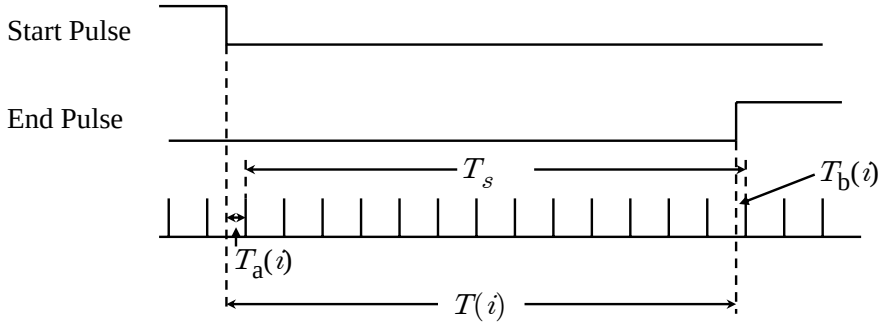


FIGURE 1.12: Time-stamping concept used in Nutt's Digital Time Intervalometer [2].

where $T_a(i)$ is time between the start pulse and the next oscillator pulse. Similarly, $T_b(i)$ is the time between the stop pulse and next oscillator pulse, and T_s is the fixed nominal time interval. In [24–26], Kalisz *et al.* digitized and extended Nutt's work, by implementing it on a micro-controller. Kalisz *et al.,* in [24], discussed many errors, such as non-linearity and jitter in time stretchers, input signal noise error, timing error, clock reference error, and the quantization process, related to Nutt's intervalometer, and proposed a model of total measurement uncertainty. In addition, for more accurate position/velocity measurement, it is shown in [24] that time-intervals can be measured with a resolution of down to 1ps. Based on this principle, many tachometry techniques for precision position/velocity/acceleration measurement have evolved [22, 23, 27]. They have also been used for the calibration of encoders in [4, 5, 28].

1.5.3 CSDT method

The Constant Sample-time Digital Tachometer (CSDT) method, proposed by Kavanagh [22], is used for the calculation of motor shaft velocity with high precision. It is similar to the M/T method [21] and even bears similarities to the work of Nutt on precision time measurement [2]. Similar to the CSDT method, a double-buffered method proposed by Prokin [29–31], is designed to ensure that the time-difference between the encoder edges occurring prior to successive samples is measured accurately. For this, registers and counter/timers interfaced to a PC AT bus, are utilized. A double-buffer method is further extended to measure low shaft speeds, when the number of encoder transitions per sample time is less than one. The desire for synchronous sampling increases the complexity of the method, though the effects of asynchronous sampling can be minimized via software modification. In [23], Mayrhofer *et.al.,* implemented a CSDT-like velocity measurement and motor controller system using a FPGA. Precise time measurement is obtained by implementing a time-stamping technique on encoder outputs. Bucchi and Landi [32] proposed a method that is similar to that of Ohmae *et al.,* applied to a linear encoder, and implemented using a micro-controller. Similar application of an auxiliary counter, if performed to estimate the shaft position at a given time (which does not necessarily correspond to a transition edge), as explained by Jian-Zhong Tang *et al.* in [33].

Both the CSDT and M/T techniques rely on an accurate timer to remove the quantization velocity error associated with the pulse count method. The CSDT

method has been shown to provide excellent average velocity information over a sample interval, along with good transient response and a wide velocity range.

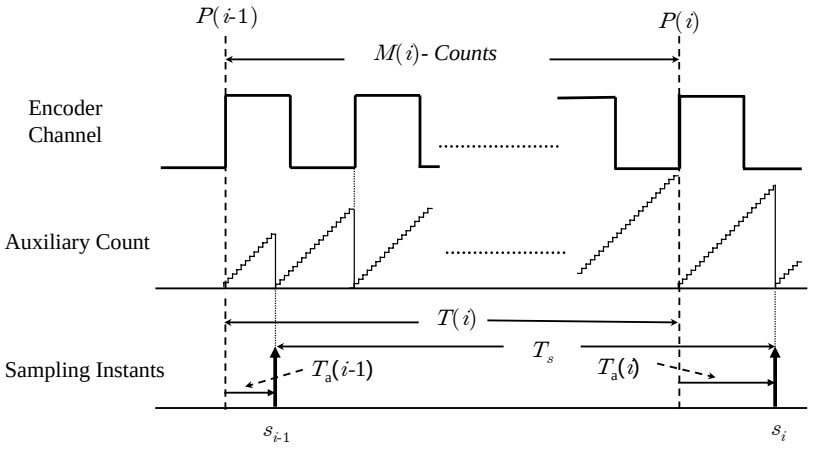


FIGURE 1.13: Timing diagram of CSDT method for measurement interval $T(i)$ (assuming measurement based on positive-going transitions only of encoder channel A or B); the digital positions at samples i-1 and i (marked as s_{i-1} and s_i) are denoted by $P(i-1)$ and $P(i)$, respectively.

A typical measurement interval is illustrated in Fig. 1.13. The average absolute shaft velocity in counts per sample-interval over the measurement interval, $T(i)$, is calculated using

$$V_i = \frac{K \cdot M(i)}{T(i)} \tag{1.8}$$

where $M(i) = P(i) - P(i-1)$, is the number of counts between transition positions $P(i-1)$ and $P(i)$, K is a constant to scale the shaft velocity to physically meaningful units, and $T(i)$ is given by

$$T(i) = T_s + T_a(i-1) - T_a(i) \tag{1.9}$$

where the sample interval, T_s, is constant and set by a microprocessor- or DSP-based controller. The auxiliary time $T_a(i)$, at the i^{th} sampling instant, is the elapsed time between this sampling instant and the most recent actual encoder transition position. The auxiliary time is measured by a high-frequency counter that is reset on each encoder transition position. Hence (1.8) can be rewritten as

$$V(i) = \frac{K \cdot M(i)}{T(i)} = \frac{P(i) - P(i-1)}{T_s + T_a(i-1) - T_a(i)} \tag{1.10}$$

where $K = 1$ when the velocity is measured in counts per sample interval.

Note that the value $M(i)$ is often termed the PCT output. It is also essential to note that the exact value of an actual encoder transition position is unknown prior to compensation, but it is approximated by the (integer) digital position $P(i)$. While implementing the quadrature decoding technique, (1.10) is modified as

$$V(i) = \frac{K \cdot M(i)}{T(i)} = \frac{P(i) - P(i-1)}{4[T_s + T_a(i-1) - T_a(i)]} \tag{1.11}$$

where $K = \frac{1}{4}$. These features ensure that the CSDT method provides a viable economical solution to high-accuracy velocity measurement in open-loop measurement systems [22]. Its closed-loop performance, which was previously considered by Kavanagh in [34] will be extended in Chapter. 4. Other FPGA- and microprocessor-based digital tachometers are described in [23, 35–39].

1.5.3.1 Low-speed CSDT

For the measurement of a shaft speed that corresponds to less than one code change per sample interval, a modified velocity measuring technique is possible, as explained in detail by Kavanagh in [22]. In this case, as exemplified in Fig. 1.14, when the auxiliary time at the i^{th} sample $T_a(i)$ is known and $T_a(i-1)$ is unknown, $T(i)$ is calculated using the last valid count $T_a(i-n)$, where n is the number of samples since the previous position change occurred. The modified average shaft velocity, at low speeds, can be written as,

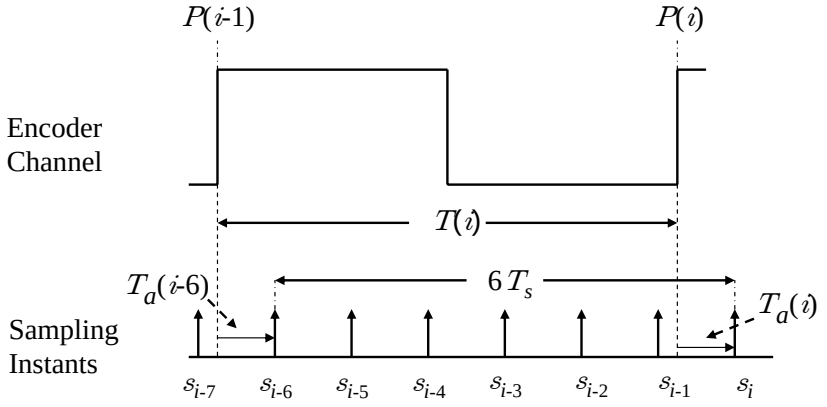


FIGURE 1.14: Timing diagram of low-speed CSDT method for measurement interval $T(i)$ (assuming measurement based on positive-going transitions only of encoder channel A or B); the digital positions at samples i-6 and i (marked as s_{i-6} and s_i) are denoted by $P(i-1)$ and $P(i)$, respectively. Note, $n = 6$ is assumed in this example.

$$V(i) = \frac{K \cdot M(i)}{T(i)} = \frac{K(P(i) - P(i-1))}{nT_s + T_a(i-n) - T_a(i)} \tag{1.12}$$

where, as before, K is a constant to scale the shaft velocity to physical units.

1.5.3.2 Oversampled constant sample-time digital tachometer (OCSDT)

In [40], Kavanagh proposed a new velocity measuring technique, the OCSDT. This technique is similar to the Oversampled Digital Differentiator (ODD) proposed in [34]. In the CSDT method, quantized position (pulse-count) and auxiliary counter/timer information is retrieved at the end of a sampling interval. However, in the OCSDT this information is obtained at a much higher rate (oversampled) during a given sample interval.

$$V(i) = \frac{K \cdot M^o(i)}{T(i)} = \frac{K \cdot M^o(i)}{nT_s + T_a^o(i-n) - T_a^o(i)} \tag{1.13}$$

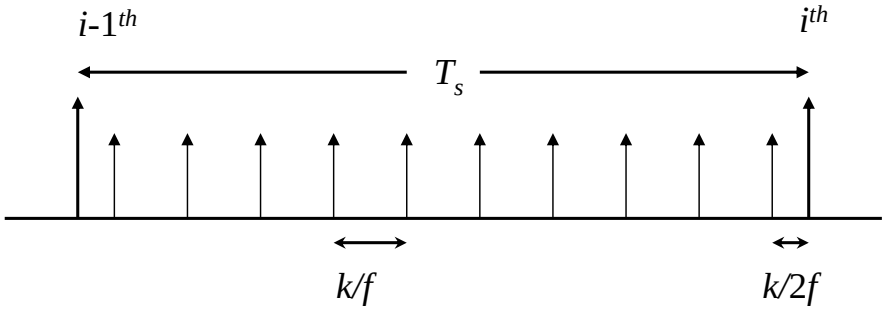


FIGURE 1.15: Time diagram showing the sub-sampling during one sampling interval.

where M^o and T_a^o are average pulse count and auxiliary time over a sub-sampling interval. Fig. 1.15 shows the sub-sampling of OCLK signals during one sampling interval, with FPGA clock frequency and k a proportional constant. As the number of encoder transitions per sampling interval is small at low-speeds, the advantages of oversampling are relatively low. At very low-speeds, where the shaft rotates at less than one pulse per sample (pps), the effectiveness of the OCSDT will be

similar to that of the CSDT, as both methods make use of the same amount of updated information from the position sensor.

As the CSDT method can provide excellent average velocity information over a sample interval, along with good transient response and a wide velocity range, this technique is selected for measuring the shaft speed in this project. The compensation information that is used to improve the performance of the CSDT could also be used to improve the OCSDT, but the advantage to be gained from the oversampling is likely to be small, so that the greater complexity involved with combining the two techniques would be hard to justify.

1.6 Previous and Proposed Work on Compensation of Encoder Outputs

Over the years, many different methods have been proposed to compensate for the position sensor non-linearities used in different applications. In order to compensate for the non-idealities in raw photodetector signals of a sinusoidal encoder, Kahl [41] proposed that precision measurement equipment be incorporated with some complex software and a learning mode. In the 'learning mode', the motor shaft is rotated at near constant speed, and the digital encoder data is stored. Using an interpolation technique, an inverse transmission characteristic is derived to generate a look-up table. This look-up table helps in measuring average shaft velocity with shorter sample time, hence giving good transient performance and compensating for the errors due to encoder non-idealities. Significant improvements are observed

in the velocity measurements, at the cost of additional hardware. Though the stability range of a modified servosystem is improved, no guarantee of stability (at lower speeds) is observed in a closed-loop configuration. Due to the very short sample-time required for this method, the neccessary high-speed microprocessor can increase the cost of the digital tachometer.

In work on the calibration of low-cost, resolver-based absolute shaft encoders, Kaul *et al.* [42] utilized a precision indexing table, so that the actual sensor position is known with high accuracy. They confirmed that the encoder inaccuracies are predominantly systematic in nature, and so can be reduced through compensation. They reported a fourfold reduction in sensor position error. This highlights a crucial assumption on which the work of this thesis is based: that a significant proportion of each encoder line error is fixed, or little changing, so that the actual encoder transition errors can be directly related to the consequent apparent velocity error.

It is also possible to derive a compensation table by coupling the incremental encoder-under-test with a high-resolution reference device, as described by Merry et al. in [4]. That paper describes a time-stamping technique using a high-resolution clock for accurate measurement of encoder events. These events relate to the pulse counter values and the corresponding encoder transition instants. These are used to determine the encoder non-idealities via a low-order least-squares polynomial fit (extrapolated to the desired (sample) time instant). Though it showed an 87% improvement in position measurement, it is necessary to run the shaft at constant speed during the learning mode and the additional high-resolution sensor

used will increase the cost of the setup.

Similarly, Merry *et al.* have described the same concept in [5] in order to compensate for errors in the calculated shaft velocity and acceleration. Improvements of 57% in velocity measurement and 92% in acceleration measurement are reported. Mancini *et al.* [43] have made the assumption that the apparent high-frequency components of position error in an encoder-based system within a large telescope cannot have a physical basis, and must be due to encoder error. The work reported in Mancini's paper concentrates on higher-frequency effects as being the ones most responsible for resultant system errors. That is particularly true in a velocity servosystem where accurate velocity-, rather than position-control, is of principal interest.

The assumption of constant, or at least smooth, shaft velocity has been intrinsic to all learning modes used in the compensation of sinusoidal, [44–47], or square encoders. A probabilistic learning technique is proposed by Kavanagh in [44] to compensate for non-idealities in a sinusoidal encoder. During the learning mode, a code-density array is obtained, which can be utilized to compose a compensation function. Using this compensation function, unsegmented, and segmented compensations are performed (i.e. using different compensation functions over different parts of the mechanical revolution).

In general, it has been shown by the researchers cited above that the line code error is the predominant form of error related to the encoder (except in crude PCT-based systems). The systems described by Merry *et al.* [4, 5], where compensation is shown to lead to a position error reduction of approximately 80%, is closest in

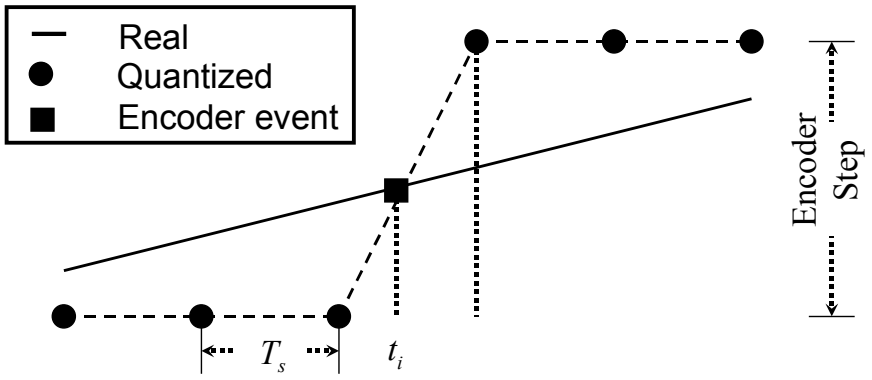


FIGURE 1.16: Concept of time-stamping used by Merry to calculate the quantized time at a sample instant to determine actual physical position(unquantized).

aim and method to that of this thesis. The time-stamping concept employed to estimate the shaft position from quantized position information is shown in Fig. 1.16. However, there are significant differences in the implementations: the work in this thesis involves calculation of a velocity reference from the time-stamped data, rather than using a reference sensor to generate a position reference. In this thesis, the data availability is limited to that obtained from the CSDT, which can be implemented on a simple micro-controller, rather than employing more specialized time-stamping hardware. Additionally, in this work consideration is given to learning over variable velocities, and for stand-alone operation.

In [48], another shaft velocity measuring technique, called the SR method, is proposed by Hachiya and Ohmae to avoid the inherent slit-error problem. This method involves calculation of the average speed of a shaft by considering the time difference between the arrival of the present encoder position and the same position of the previous revolution, as shown in Fig.1.17, where T_{d1} is the time between the present and immediately previous encoder transition positions (Position

1). Similarly, for position i, time differential will be T_{di}. Though this technique compensates for the influence of the slit error, the output shaft velocity is delayed by one revolution, leading to a very slow response. This drawback makes the solution less attractive.

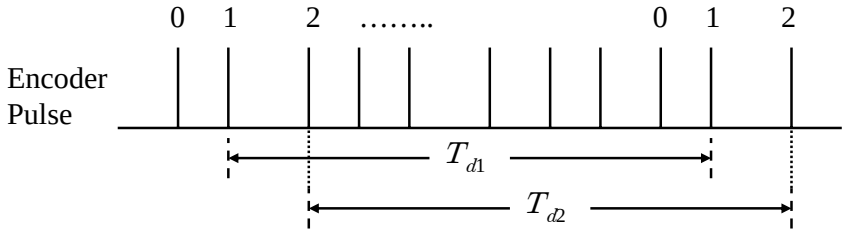


FIGURE 1.17: Timing diagram of SR method showing the time determination for calculating average shaft velocity (times displayed are based on measurement of positions 1 and 2.

Similar work on sinusoidal encoders was performed by Tan *et al.* using neural networks to derive a look-up table of encoder errors [49].

1.7 Hardware Implementation

Fig. 1.18 shows a typical block diagram of a closed-loop electrical drive system which includes a CSDT-based digital tachometer. The digital tachometer utilized in this work consists of an incremental square-wave encoder, coupled to an electrical motor, to detect the position information of the shaft, and an FPGA to process the data obtained from the encoder and to measure time and position information for the DSP. Using this processed data obtained from the FPGA, the DSP implements the compensation and closed-/open-loop control algorithms.

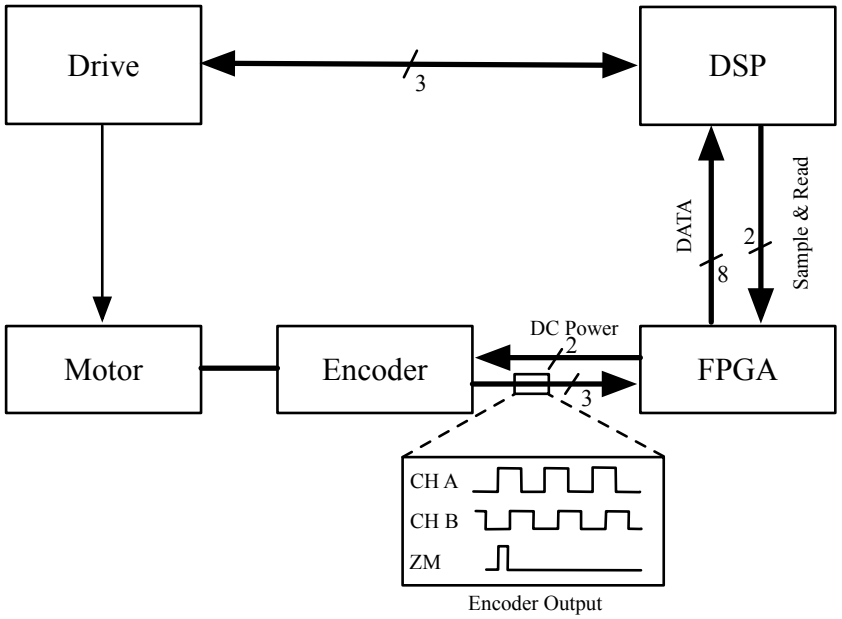


FIGURE 1.18: Block diagram of a typical FPGA/DSP-based servosystem controller.

1.7.1 Optical encoders used in experimental work

For this work, three different optical incremental square-wave encoders, with different resolutions and from different manufacturers, are tested. Encoders with a resolution of 200 ppr from HP [50], 300 ppr from OVW [51], and 360 ppr from Omron [52], are selected. The use of encoders of different quality/price and type (two of the encoders are coupled to the motor shaft, while the third is of the hollow shaft type whereby the codewheel slips onto the motor shaft) facilitates testing, as it indicates the generic nature of the algorithms proposed. The details of these encoders are shown in Table 1.1

TABLE 1.1: List of selected encoders that are utilized for experimental analysis

Factors	Encoder A	Encoder B	Encoder C
Type	Incremental	Incremental	Incremental
Manufracturer	Omron	OVW	HP
Resolution	360	300	200
No of Channels	3	3	3
Shaft Mounting	Coupling	Coupling	Hollow

1.7.2 Field Programmable Gate Array

A field-programmable gate array (FPGA) is an economical way of implementing complex custom hardware on a single chip. The FPGA has evolved from a wide variety of programmable logic devices (PLDs) which were employed principally as replacements for 'glue logic', small-scale-integration (SSI) parts. The need for application-specific integrated circuits (ASICs) [53] led to the development of the gate array in which connections between an array of pre-configured logic blocks can be implemented in silicon through use of an application-specific interconnect layer. In the next generation, a re-programmable set of interconnects replaced the pre-configured logic blocks to permit very fast design cycles, with particular advantages in a prototyping environment [54, 55]. Typically, FPGA logic blocks can be as simple as a transistor or as complex as a microprocessor, and are well capable of implementing different sequential and combinational logic functions.

A low-cost Xilinx FPGA device was selected. Xilinx, currently the world's largest FPGA manufacturers, were the first to introduce these devices in the mid-1980s. The Xilinx logic blocks are more complex than those of their competitors and have great flexibility, but there is a possibility of under utilizing on-chip resources. The

FPGAs produced by Xilinx Inc. are broadly classified into two families: Spartan and Virtex. The Spartan-FPGAs are designed for high-volume production, where low-cost and easy implementations are major constraints, while Virtex FPGAs are aimed at high-performance systems.

1.7.2.1 The Xilinx XC4000 FPGA

A simple (and effectively old-fashioned relative to the present Spartan-6 series) Spartan Xilinx XC4000 series FPGAs [56–58] is selected for this work. This proved capable of implementing designs of the complexity required for the task being described. The product introduced in 1990, was available for US $10 (in large quantity) and slightly scaled-down versions of these devices [59] increased the usage of FPGAs in wide-spread applications, due to the XC4000's cost-effectiveness and flexibility in implementation.

The configurable logic block (CLB) is the principal element in an FPGA, determining its processing capability. The easily programmable CLBs are arranged in a two-dimensional array form. They are interconnected by a powerful hierarchy of versatile routing channels. The selected FPGA, XSC10, is built with a 14 by 14 CLB matrix, as shown in Fig. 1.19. A simplified block diagram of a CLB, adopted from the Xilinx data book, is shown in Fig. 1.20. Each CLB contains three Look Up Tables (LUT) or Function Generators (FG), where FG4s can implement any logic functions of four variables, two flip-flops and two groups of signal-controlling multiplexers. Using the optional modes in CLBs, functional generators (F-LUT and G-LUT) can also be used as random access memory (RAM), because read and

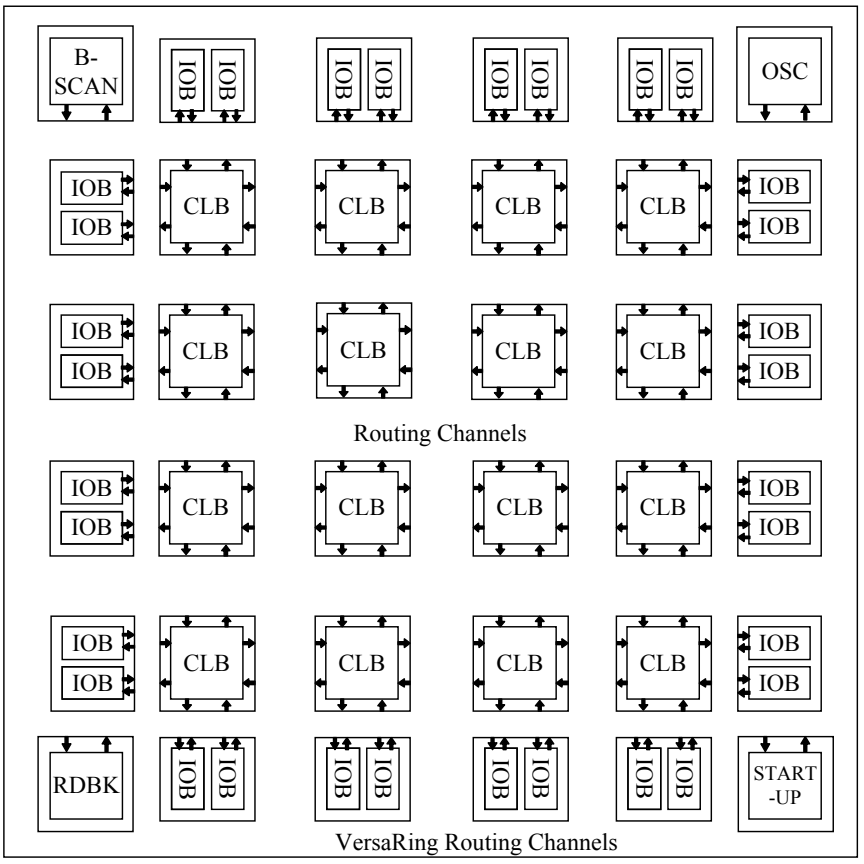


FIGURE 1.19: Block diagram of a basic FPGA (Extracted from Xilinx data book).

write operations can be performed significantly faster than off-chip implementations. The presence of the other function generators allows more complex logic functions to be performed, with outputs from FG4s and an external input. In addition to external input signals, the outputs from FGs are connected to flip-flops. A detailed description of CLBs is available in [56–58]. Additional features of the XC4000 series include:

- *Input/Output Block (IOB)*: The Input/Output Blocks, which are user configurable, provides the interface between the FPGA and external logic signals. Each IOB can be configured for input, output, or bidirectional signals.

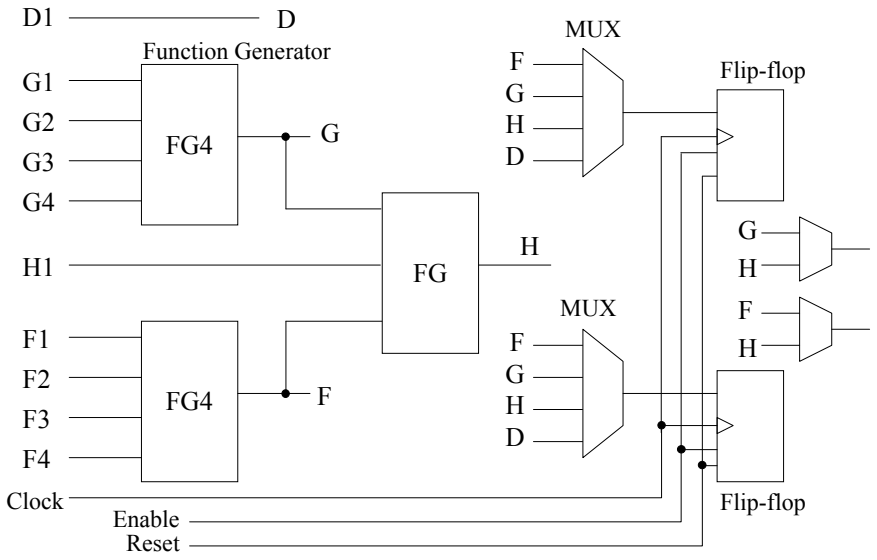


FIGURE 1.20: Simplified logic diagram of Configurable Logic Block (CLB) (slightly modified extract from Xilinx data book).

These high speed IOBs are capable of a number of programmable features such as optional pull-up or pull-down, low speed or high-speed, possibility of registered input and/or output, etc.

- *Routing and programming*: The CLBs, which are used for most of the logic implementation in FPGAs, are encircled by the routing channels providing the connection between inputs and outputs of the CLBs and IOBs. Though direct interconnections between adjacent CLBs are possible, many connections are provided via routing channels and controlled by routing switches. In complex designs, the latter option is subjected to considerable propagation delays, so that dedicated high-load lines with high-drive clock buffers are available for clock signals, ensuring the existence of minimal skew from such signals.

Distributed static-RAM cells situated adjacent to the programmable resources of the device are used to configure the FPGA. By application of a serial configuration bit-stream, clocked into the FPGA from an EPROM or another data bus, these cells can be programmed.

- *Software Tools*: The FPGA-based design process consists of many stages. The designed circuits can be implemented either using a schematic approach or VHDL tools. In this project, Viewlogic schematic capture software is adopted for circuit design purposes. The desired schematic model can be built with the help of a library of pre-designed logic elements such as counters, arithmetic-logic-units, etc. The software provided by Xilinx facilitates routing, placement, translation, and design-rule checking of the required circuit.

1.7.3 dSPACE DS1104

The dSPACE (ds1104 board produced by dSPACE GmbH, Paderborn, Germany,) is a data-acquisition system combined with an independent processing system to implement digital control models. It is intended to be a complete real-time control system based on a PowerPC (PPC) 603e fixed-point processor [60]. In this project, the DS1104 is used for learning the slit error associated with a particular encoder and for implementation of the sensor compensation and servosystem measurements and/or control required in this project. In addition to the PPC (the master processor), the DS1104 R&D Controller board, shown in Fig. A.1 of Appendix A, includes a slave DSP, interrupt controller, memory, host interface,

and timers. The master PPC runs at 250 MHz. It controls two ADC units (a four channel 16-bit analog-to-digital Converter (ADC) unit and a four channel 12-bit ADC unit), a 16-bit digital-to-analog Converter (DAC) unit, incremental encoder interface, 20-bit digital I/O and a serial interface (RS-232). The incremental encoder interface can take inputs from two digital incremental encoders and has 24-bit position counters. Both digital incremental encoder channels can handle encoder signal frequencies of up to 1.65 MHz. Note, however that the DS1104's encoder interface circuitry is not used in this work, being replaced by the more flexible and powerful FPGA-based circuitry.

The slave-DSP subsystem is based on a TMS320F240 DSP micro-controller which can be used for advanced I/O purposes. The *ControlDesk* software provided with the DS1104 is used to design the system implementation and interface for the DS1104 board. The software interface also provides various useful features, including linkages with MATLAB and an oscilloscope-like variable tracing capability. An I/O breakout box is used to provide connectivity between the FPGA board and the dSPACE card for transmitting raw encoder position and time data.

1.7.4 Electrical drives used in experimental work

Two different experimental setups are considered to test and analyze the encoder behaviour and its influence on the outputs. One electrical drive is mounted with a very heavy fly-wheel which will assist in providing very smooth velocity output. A BRU-200 brushless electrical drive [61] from Electrocraft is coupled with a heavy inertia (fly-wheel) to provide smooth velocity. The BRU-200 was sold as a

high-performance sinusoidal brushless industrial drive. The drive module provides control and power to S-series permanent magnet synchronous motor (S-3007). The control circuit of the BRU-200 drive includes a 16-bit microprocessor and personality module which allows matching with the motor, thereby facilitating tuning and setup. A standard RS-232/RS-422 serial interface is used to connect a PC to the drive, to modify tuning, change limit values, or to monitor variables/status in the servosystem. A standard decimal line count encoder, which is factory installed on the motor, is used for commutation purposes. However, with the drive controller in current-loop mode, velocity feedback utilizes the digital tachometers described in this thesis when implementing closed-loop control.

The second experimental setup is lightly loaded, enabling the motor to run at high speeds and to implement closed-loop control. An Aerotech 4020 linear servo amplifier is used to analyze encoder performance and test the learning algorithm at medium and high speeds. The power output stage of this servoamplifier consists of NPN power transistors which operate in the linear mode. The servo controller includes a 741-type pre-amplifier with position and rate compensation, driving a power amplifier configured in a current feedback mode. This pre-amplifier features three adjustable scale factors for the inverting input: balance adjustment, a non-inverting input and a gain adjustment for lag-lead compensation. The Aerotech 4020 servosystem powers and controls an Aerotech 1050DC (a 1000 series) brush-type rotary DC servomotor [62] which features a dynamically-balanced skewed rotor and tachometer for analogue velocity feedback. However, in-built encoders (in both drives) are not used for testing.

TABLE 1.2: Electrical drives selected to build the servo system

Specifications	Aerotech1050	BRU 200 (S-Series)
Maximum Speed (rpm)	5000	5000
Continuous Stall Torque (N-m)	0.35	0.79
Peak Torque (N-m)	2.52	-
Rated Power (W)	146	-
Bus Voltage (VDC)	40	125-375
Torque Constant (N-m/Amp)	0.07	0.28
Encoder resolution (counts/rev)	-	2000

1.8 Aim and Objectives

The fundamental goal of this research is to improve the performance of high-bandwidth open-loop angular velocity measurement systems, or closed-loop servo systems, using the constant sample-time digital tachometer (CSDT) method, and particularly to design a new improved technique to compensate for high-frequency sensor errors.

1.8.1 Objectives

The primary objective of the thesis is therefore to propose new techniques to obtain improved tachometer output with no, or minimal, additional hardware and cost, by compensating for the high-frequency error due to non-idealities in the code-wheel of an encoder.

The augmentation of a drive system by encoder compensation can be divided into a number of linked sub-problems:

- Calculation of an uncompensated velocity estimate from the time-stamped data associated with the CSDT.
- Generation of 'reference' (i.e. improved) velocity or position estimates through filtering or curve-fitting techniques, which when compared with the initially calculated velocity, provides some information on the encoder error.
- Processing this data over many samples to generate a look-up compensation table of code errors (transition location errors), and
- Use of this table during the subsequent operational phase of the motion system.
- Development and analysis of a mathematical and MATLAB-based model for the proposed technique.
- The proposed compensation algorithms are implemented experimentally in both open-loop and closed-loop systems.
- The results obtained on systems employing different encoders and drives are designed to verify the benefit to be obtained by implementing the new algorithms.

1.9 Outline of the Thesis

Chapter 2: The chapter introduces three proposed learning techniques, and presents details of how the algorithms work.

Chapter 3: The chapter contains an analysis of the error sources associated with

encoders and digital tachometers, and of the error reduction due to the use of compensation algorithms.

Chapter 4: The proposed learning techniques are simulated using MATLAB and the results are presented and discussed. This chapter also consists of a detailed description of the experimental implementation of the learning methods in open-loop and closed-loop systems. Results obtained from different encoders are compared and analyzed.

Chapter 5: Finally, conclusions are drawn, based on simulation and experimental results. Future work and limitations of the proposed method are also presented.

Chapter 2

Description of the Learning Algorithm

2.1 Introduction

This chapter is dedicated to the determination of a suitable statistical method to determine the slit errors. The methodologies and the optimal usage of these selected procedures are explained. It is necessary to determine the unknown slit errors, δ_i, and interval errors, $\delta_{i,i+1}$, of the encoder code-wheel. Approximate linear equations can be formulated which are dependent on the line errors of a particular encoder. Solution of these equations should allow estimation of the encoder errors. However, these mathematical models cannot usually be solved explicitly, and numerical methods to obtain approximate solutions are needed. Depending on the number of constraints, such as the required accuracy, the errors involved,

the speed and time of convergence, and the memory consumption required, suitable numerical methods can be adopted. These numerical methods can be broadly classified as direct and iterative (indirect) methods.

Direct methods, such as Gaussian eliminations, QR factorization, etc., are primarily adopted to determine the exact solution after a finite number of operations for a linear system. However, these methods only lead to a theoretically exact solution in a finite number of steps. In practice, there are errors in the computed value due to rounding errors in the computation, arising from the finite length of numbers in standard computer arithmetic. In contrast, the iterative methods, first introduced probably by Gauss, provide solutions for both linear and non-linear equations. These indirect methods are simple and require less computer storage. They can also provide solutions in a simpler way for special structures like sparse matrices (for which memory storage requirements can be significantly reduced) and are suitable if there are many unknown variables. As against the determination of exact solutions, these methods tend to calculate approximate solutions, with successive approximations of increasing accuracy, within acceptable boundaries.

The main disadvantages of iterative methods are their relative slow convergence and, at times, their undesirable divergent behaviour [63, 64]. It can be observed from the graph in Fig 2.1, reproduced from [3], that solutions for a problem can typically be determined more quickly with iterative methods than with direct methods. The number of iterations required to attain the required residual value depends on many constraints including: type of the iterative method selected, errors in the data, etc. Due to their limitations, direct-method type techniques

are not considered for the algorithms which are designed to learn the slit errors. The iterative methods are typically classified as

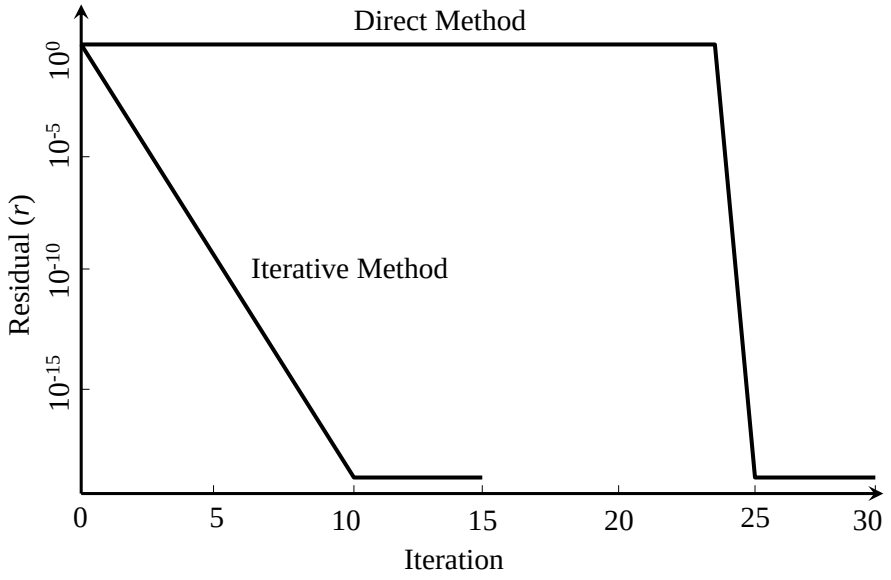


FIGURE 2.1: A graph showing the calculated residual in a system against the number of iterations (reproduced from [3]).

- Stationary, or classical, iterative methods, and
- Non-stationary iterative, or Krylov subspace, methods.

The Jacobi, Gauss-Seidel, and successive over-relaxation techniques are some examples of stationary methods, while conjugate gradient, generalized minimal residual, bi-conjugate gradients, etc., are Krylov subspace type methods. In the field of sensor networks, numerical analysis is widely used to determine the exact/true value of an acquired signal that is affected by noise. Down the years, many new analytical methods have been derived from the primary methods. In [65], Barooah *et.al.*, proposed two optimal estimate algorithms, based on the Jacobi method, to compute the vector-value variables at the nodes in a sensor network affected by uncorrelated zero-mean noise. One algorithm is aimed at a situation without failures,

while the other is chosen to deliver robust performance even with temporary communication failures. Though these iterative algorithms are simple and robust, the convergence to correct values is slow. Later in [66, 67], the same authors proposed a new algorithm, the overlapping subgraph estimator algorithm (OSE), which is distributed and robust to temporary communication faults, and is executed asynchronously. A new Embedded Polygons Algorithm (EPA) as proposed by Delouille et.al., in [68] is a combination of the Jacobi and embedded tree methods, which is essentially a block-Jacobi iterative method, for obtaining least-mean-squares error estimation of node variables in a distributed manner. In [66, 67], a comparative study is performed between Jacobi, modified EPA (adapted from EPA in [68]), and OSE methods to evaluate average energy consumption and the speed of convergence. It is observed that OSE needed relatively far fewer iterations to converge then other methods, and consumes much less energy. However, these methods are relatively complex to implement. For overdetermined systems, as in this thesis, the least-squares approximation method is widely used because of its simplicity, effectiveness and completeness [3]. Nonetheless, its main disadvantages are its sensitivity to outliers and possibly its poor extrapolation property over long ranges. Though most of these methods can be adopted to the problem under consideration, consideration of the limitation imposed by time- and memory-based constraints and the desire for an easily coded algorithm, the following three methods are considered for developing the learning algorithm to compute position/slit errors in encoder code-wheels:

1. Pseudoinverse-A (based on calculation of slit errors)

2. Pseudoinverse-B (based on estimation of interval errors), and

3. An iterative method

The final algorithm is based on a simple novel and intuitive iterative methodology, while the remaining two methods (Pseudoinverse-A and Pseudoinverse-B) are based on the least-squares solution of linear equations. The different algorithms are recommended to facilitate different types of implementation, based on memory space and convergence speed requirements. These algorithms utilize the raw position and time data from an incremental encoder and should be capable of predicting these position/slit errors in the code-wheel, without any prior knowledge of the velocity profile, or any reference encoder/hardware. Hence, additional hardware, and maintenance costs will be avoided.

Because no reference encoder is used in the work reported, some small velocity variations that are correlated to shaft position are inevitable. One consequence of this is that a low-frequency component of the code wheel error over the circumference, or a non-orthogonality in the code wheel setup [1], may not be easily distinguished from a position-related velocity variation. However, very-low-frequency position error components are inconsequential for the accuracy of a velocity servosystem, due to the negligible measured velocity error caused by the differentiation of the position error. Such integral-type position errors can be measured (e.g. as by Merry *et al.*[4]) if necessary, through use of a huge inertial load, or a reference sensor. The effect of these very-low-frequency positions errors are studied later when considering the simulation model of the encoder described in Section. 3.4. Other low-frequency effects such as friction do not greatly influence the learning process.

The techniques described in this thesis, which automatically estimate any uncorrelated line position errors, are intended to provide the required improvement in velocity control. In the next section, a detailed description of the code-wheel error and nomenclature is presented. In subsequent sections, two pseudoinverse-based methods: Pseudomethod-A and Pseudomethod-B, intended for off-line implementation, are proposed for determining the slit error and the interval error. For on-line implementation, an iterative method, which consumes less memory space, is presented in Section 2.4.

2.2 Code-wheel Error - Description and Nomenclature

Before proposing the adopted learning algorithms, it is essential to understand the nomenclature of a typical incremental shaft encoder. For illustrative purposes, a non-ideal code wheel with eight counts per revolution, $L = 8$, is shown in Fig. 2.2, where P_i, $i = 1, \ldots, 8$, are the ideal encoder transition positions over the disk circumference (P_0 coincides with P_8), the corresponding actual positions are $\bar{P}_i$, and their slit errors are denoted by δ_i, so that $\bar{P}_i = P_i + \delta_i$. For convenience, disk rotation in the clock-wise direction is considered as a positive velocity. The slit errors are considered negative when the actual transition position is less than the ideal transition position, and positive otherwise. For example, if P_{12} represents the distance between ideal encoder transition positions P_1 and P_2, the distance

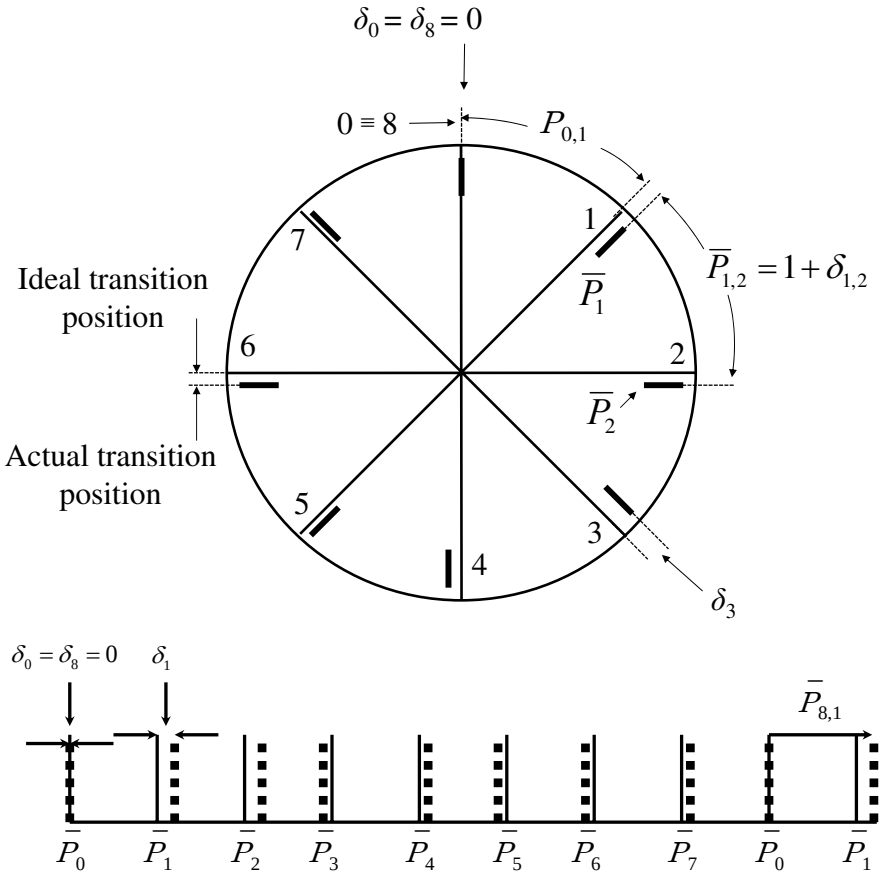


FIGURE 2.2: Non-ideal code wheel with eight counts per revolution showing ideal (integer) and actual transition positions, where position 0 (which corresponds to the encoder's zero marker) provides the reference.

between non-ideal encoder transition positions 1 and 2 is given by

$$\bar{P}_{1,2} = \delta_2 + P_{1,2} - \delta_1 = P_{12} + \delta_{1,2} \tag{2.1}$$

where $\delta_{1,2}$ is the corresponding interval error. Because of these non-uniform pulse widths over the circumference of the code wheel (which exist in all encoders), the shaft velocity estimated by the CSDT will contain high-frequency errors, i.e. the ability of the CSDT to measure the velocity with extremely high accuracy is compromised. Given the smoothness in actual shaft velocity that is physically imposed by the combined inertia of rotor and load, an assumption can be made

that much of the apparent high-frequency content in the CSDT output is due to the aforementioned encoder error. By isolating this high-frequency content and using it to derive estimates of the errors in encoder transition locations, the ability to accurately measure the kinematic parameters of the system is greatly enhanced.

2.3 Description of the Pseudoinverse-Based Learning Algorithms

The compensation for slit error in an encoder code-wheel can be provided either by calculating slit error or interval error. For this, two algorithms (Pseudoinverse-A to determine slit error and Pseudoinverse-B, a slightly modified version of Pseudoinverse-A, for calculating interval error), based on pseudoinverse/least-square principles, are proposed.

2.3.1 Description of Pseudoinverse-A method

It is assumed that N_s samples of the CSDT data (digital positions, P, and auxiliary times, T_a, from which CSDT velocity can be calculated, as required), are stored prior to compensation, where $N_s > L$, the number of encoder lines. At sample i, the CSDT velocity estimate, $V(i)$, will be noisy due to transition location error. Therefore, one cannot assume that the actual velocity is precisely known, though inertial considerations imply some smoothness in the shaft velocity. An improved ('reference') velocity estimate, $V_r(i)$, can be calculated (off-line) by use

of a high-order, zero-phase, Butterworth filter [69] that more accurately estimates the average velocity over measurement interval i. (When velocity is changing predictably, e.g. approximately following a smooth curve, preprocessing via curve fitting can also assist in the calculation of V_r). In addition, the reference velocity can also be estimated using other simple techniques like averaging a number of previous velocity samples, or implementing interpolation or extrapolation algorithms using curve-fitting techniques of different degrees. Consideration is given in Section 4.2.3 to the choice of methods for the computation of reference velocity estimates that facilitate real-time implementation, the methods being described in detail in that section. The generation of V_r is intrinsic to the operation of the learning algorithm because the difference between the CSDT-derived velocity and V_r allows estimation of the encoder line errors. For generating a reference velocity using a filtering technique, the CSDT velocity output, which is computed using raw position, P, and time, T_a, data is used as an input to the filter. A low-pass filter of suitable bandwidth can be selected to separate high-frequency noise from the CSDT velocity. In order to avoid any lag in the output reference velocity, V_r, a zero-phase filter can be utilized when the process is not being implemented in real-time, i.e. for off-line operation. Hence, V_r contains much lower high-frequency error than that associated with the raw CSDT output.

If the actual locations used in the CSDT calculation are $P(i) + \delta_{P(i)}$ and $P(i-1) + \delta_{P(i-1)}$, where, in general, δ_k represents the line position error at code k, so that $\delta_{P(i)}$ is the encoder line error associated with the digital position at sample i. The reference average velocity $V_r(i)$, which will be close to the actual velocity

over measurement interval $T(i)$, is

$$V_r(i) = \frac{(P(i) + \delta_{P(i)}) - (P(i-1) + \delta_{P(i-1)})}{T(i)} \tag{2.2}$$

when the reference velocity estimate, $V_r(i)$, is indeed correct. For positive velocity, one can write

$$\delta_{P(i)} - \delta_{P(i-1)} \approx V_r(i)\,T(i) - P(i) + P(i-1) = e(i) \tag{2.3}$$

where the equality has been replaced by an approximation in recognition of the fact that the differential line error, $e(i)$, the difference in the errors associated with two (not necessarily adjacent) encoder transition errors is noisy because of unmodelled encoder non-idealities, and because of any error in V_r.

When the position moves through the zero marker in a given sample interval, (2.3) is replaced by

$$\delta_{P(i)} - \delta_{P(i-1)} \approx V_r(i)\,T(i) - P(i) - L + P(i-1) = e(i) \tag{2.4}$$

However, because the zero marker provides the position reference, one can assume that $\delta_0 = 0$, giving a linear equation in only one variable when $P(i) = 0$ or $P(i-1) = 0$.

Similarly, in the case of negative velocity, when the shaft direction is anti-clockwise, (2.2) is re-written as

$$V_r(i) = \frac{(P(i-1) + \delta_{P(i-1)}) - (P(i) + \delta_{P(i)})}{T(i)}. \tag{2.5}$$

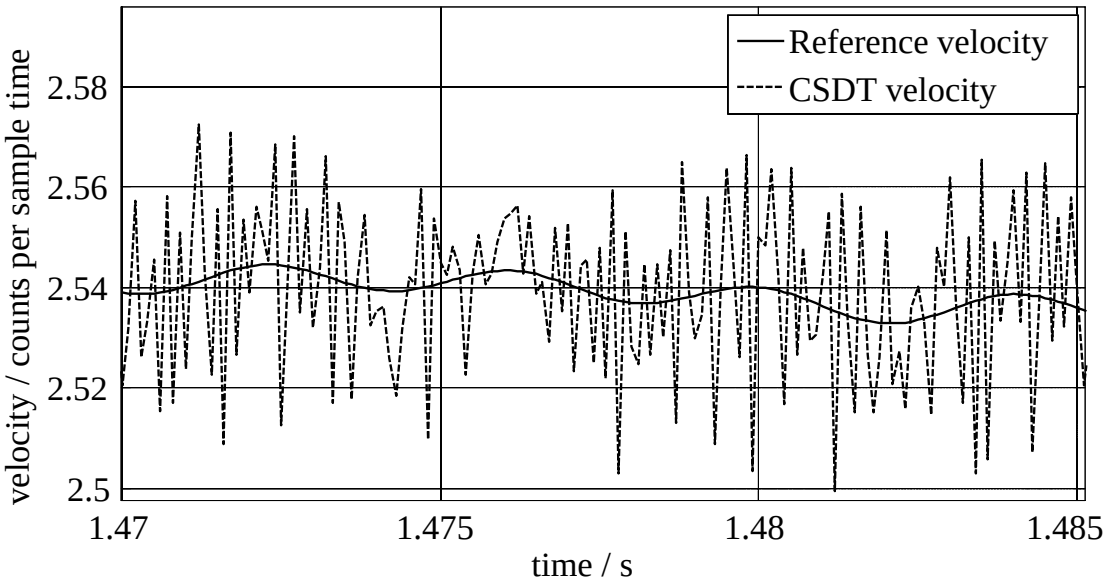


FIGURE 2.3: Comparison of CSDT velocity and the reference velocity generated using a zero-phase filter. (Example from a MATLAB-generated, simulated CSDT/encoder)

Then, the differential line error, $e(i)$, is determined as

$$\delta_{P(i)} - \delta_{P(i-1)} \approx P(i-1) - P(i) - V_r(i)\,T(i) = e(i), \tag{2.6}$$

and Eq. 2.4 remains valid for negative velocity when the position passes through the zero marker.

By storing N_s samples, a simple N_s by $L-1$ coefficient matrix equation can be established based on the matrix equation

$$\mathbf{A}\boldsymbol{\Delta}_{\mathbf{A}} \approx \mathbf{e} \tag{2.7}$$

where the $L-1 \times \mathbf{1}$ column matrix $\boldsymbol{\Delta}_{\mathbf{A}}$, and an $N_s \times \mathbf{1}$ column matrix $\mathbf{e}$, differential line error matrix, are

$$\boldsymbol{\Delta}_{\mathbf{A}} = [\delta_1, \delta_2, \ldots, \delta_{L-1}\,]^T \tag{2.8}$$

$$\mathbf{e} = [e(1), e(2), \ldots, e(i), \ldots, e(N_s)\,]^T \tag{2.9}$$

and $\mathbf{A}$ is a sparse coefficient matrix, the rows of which are mostly exemplified by

$$\mathbf{A} = \begin{bmatrix} 0 & 0 & \ldots & -1 & 0 & \cdots & 0 & 0 & 1 & 0 & 0 & 0 & 0 & \ldots \\ 0 & 0 & \ldots & 0 & 0 & 0 & 0 & 0 & -1 & \ldots & 0 & 1 & 0 & \ldots \\ & & & & & & \vdots & & & & & & & \end{bmatrix}. \tag{2.10}$$

When the sensor output detects negative velocity, the matrix $\mathbf{A}$ will typically be of the form,

$$\mathbf{A} = \begin{bmatrix} 0 & 0 & \ldots & 1 & 0 & \cdots & 0 & 0 & -1 & 0 & 0 & 0 & 0 & \ldots \\ 0 & 0 & \ldots & 0 & 0 & 0 & 0 & 0 & 1 & \ldots & 0 & -1 & 0 & \ldots \\ & & & & & & \vdots & & & & & & & \end{bmatrix}. \tag{2.11}$$

An estimate of $\mathbf{\Delta_A}$ that is optimum in the least-squares sense can be obtained by

$$\mathbf{\Delta_A} = \mathbf{A}^{+}\mathbf{e} \tag{2.12}$$

where $\mathbf{A}^{+}$ is the pseudoinverse of $\mathbf{A}$. The errors in the encoder circuitry have been found to be well approximated by the deterministic line error at each encoder position, plus a smaller white noise component. The latter is effectively removed by (2.12). To aid understanding, a summary of the Pseudoinverse-A algorithm is listed in sequential steps in Table 2.1.

2.3.2 Description of Pseudoinverse-B method

For calculating interval errors, $\delta_{i,i+1}$, of a code-wheel, the Pseudoinverse-A method can be adopted, with slight modifications. The determination of interval error, $\delta_{i,i+1}$, is relevant as it has been observed that the condition number relating to the coefficient matrix can be improved if the encoder error estimation problem is reformulated in terms of the interval width errors, i.e. the errors in the distance between adjacent transition positions (as will be demonstrated in Chapter 4). Using $\delta_{j,(j+1)} = \delta_{j+1} - \delta_j$ so that $\delta_k - \delta_j = \delta_{j,(j+1)} + \delta_{(j+1),(j+2)} + \cdots + \delta_{(k-1),k}$, one can replace (2.2) by

$$V_r(i) = \frac{P(i) - P(i-1) + \delta_{P(i-1),P(i)}}{T(i)} \tag{2.13}$$

where $\delta_{P(i-1),P(i)}$ is the summation of interval errors between $P(i-1)$ and $P(i)-1$.

TABLE 2.1: Sequence of operations performed to implement the Pseudoinverse-A learning algorithm

Pseudoinverse-A method
Initialization : $\delta_i = 0$; where i is an integer in the range $0 \leq i < L$
1. Encoder digital position $P(i)$ and auxiliary time $T_a(i)$ are obtained over N_s samples.
2. Average velocity over a sample interval, $V(i)$, is calculated from raw encoder data using the CSDT method.
3. The reference average velocity $V_r(i)$ is predicted by performing either zero-phase filtering or curve fitting on $V(i)$.
4. Using $V_r(i)$ and $V(i)$, the differential line error (between the i^{th} and $i-1^{th}$ samples), $e(i)$, is calculated. For positive rotation where the position is not through zero (2.3) is used, positive rotation through zero uses (2.4), while negative velocity requires use of (2.4) or (2.6), as outlined in the text.
5. Each row of the sparse co-efficient matrix $\mathbf{A}$, (of size $N_s \times L-1$), is compiled by inserting -1 and 1 in the $P(i)^{th}$ and $P(i+1)^{th}$ columns and filling the reminder of the rows with 0s.
6. The pseudoinverse of $\mathbf{A}$, $\mathbf{A}^+$, is computed.
7. $\mathbf{A}^+$ is postmultiplied by the differential line error matrix, $\mathbf{e}$, to determine the slit error δ_i, where $0 \leq i < L$.

Comparing this positive reference velocity, $V_r(i)$, with the CSDT velocity, $V(i)$, the position error can be calculated as,

$$e(i) = V_r(i)\,T(i) - P(i) + P(i-1) \approx \sum_{k=P(i-1)}^{P(i)-1} \delta_{k,(k+1)}\,. \tag{2.14}$$

This formula assumes positive rotation and that the position does not pass through zero (i.e. the zero marker, ZM, is not activated over the sample. Similarly to (2.4),

when the sensor output passes through the zero marker in the clockwise direction, (2.14) can be replaced by,

$$e(i) = V_r(i)\,T(i) - P(i) - L + P(i-1) \approx \sum_{k=P(i-1)}^{L-1} \delta_{k,(k+1)} + \sum_{k=0}^{P(i)-1} \delta_{k,(k+1)}\,. \quad (2.15)$$

When the shaft rotates in anti-clockwise direction (negative direction), $V_r(i)$ and $e(i)$ are calculated as,

$$V_r(i) = \frac{P(i-1) - P(i) + \delta_{P(i-1),P(i)}}{T(i)}, \quad (2.16)$$

and

$$e(i) = V_r(i)\,T(i) + P(i) - P(i-1) \approx \sum_{k=P(i)}^{P(i-1)-1} \delta_{k,(k+1)}\,. \quad (2.17)$$

Finally, when the shaft crosses the zero-marker during the sample interval, $\mathbf{e}$ can be re-written using

$$e(i) = V_r(i)\,T(i) - P(i) - L + P(i-1) \approx \sum_{k=P(i)-1}^{L-1} \delta_{k,(k+1)} + \sum_{k=0}^{P(i-1)} \delta_{k,(k+1)}\,. \quad (2.18)$$

The corresponding matrix equation is

$$\mathbf{\Delta_B} = \mathbf{B}^{+}\mathbf{e} \quad (2.19)$$

where

$$\mathbf{\Delta_B} = [\delta_{0,1}, \delta_{1,2}, \ldots, \delta_{(L-2),(L-1)}]^T. \quad (2.20)$$

The elements of B are binary (i.e. 0 or 1), with structure of the form exemplified by

$$\mathbf{B} = \begin{bmatrix} 0 & 0 & \dots & 1 & 1 & \cdots & 1 & 1 & 0 & 0 & 0 & 0 & 0 & \dots \\ 0 & 0 & \dots & 0 & 0 & 0 & 0 & 0 & 1 & \dots & 1 & 0 & 0 & \dots \\ & & & & & & \vdots & & & & & & & \end{bmatrix}. \tag{2.21}$$

While there are L lines (and L line differences), the rank of $\mathbf{B}$, like that of $\mathbf{A}$, will be $L-1$, because

$$\sum_{j=0}^{L-2} \delta_{j,(j+1)} + \delta_{(L-1),0} = 0. \tag{2.22}$$

Therefore, for any sampling interval in which the encoder passes through the zero marker, (2.14) is replaced by $e(i) = -\sum_{k=P(i)}^{P(i-1)-1} \delta_{k,(k+1)}$ for positive rotation. As before, similar equations are used when the velocity is negative.

In a typical implementation, the condition numbers of $\mathbf{A}$ and $\mathbf{B}$ (the corresponding methods of estimating the encoder errors being termed 'Pseudoinverse A' and 'Pseudoinverse B' henceforth) were found to be of the order of 100 and five, respectively, (when the system velocity during the learning phase varied from approximately two to four codes per sample interval). Also, by using (2.19), and (2.12), the performance of these two methods will be analyzed in later chapters based on various criteria, such as convergence speed, memory required, and percentage error reduction, along with the performance when using different reference velocity generation techniques. A brief listing of the Pseudoinverse B learning algorithm is presented in Table. 2.2.

TABLE 2.2: Sequence of operations performed to implement Pseudoinverse-B learning algorithm

Pseudoinverse-B method
Initialization : $\delta_{i,i+1} = 0$; where i is an integer in the range $0 \leq i < L-1$.
1. Encoder digital position, $P(i)$, and auxiliary time, $T_a(i)$, are obtained over N_s samples. 2. Average velocity over a sample interval, $V(i)$, is calculated from raw encoder data using the CSDT method. 3. The reference velocity $V_r(i)$ which is required to calculate interval error is predicted either using the high frequency zero phase filter, or by implementing curve-fitting on CSDT velocity $V(i)$. 4. The position error between the (i-1)th and ith samples, e(i), is calculated. Use is made of (2.14), (2.15), (2.16) or (2.18), as appropriate, depending on direction of rotation and whether an interval involves rotation through the zero marker. 5. Matrix B is generated. This consists of N_s rows, where each row represents a single sample , and $L-1$ columns (L is the resolution of the encoder). Each of the rows in this matrix is compiled by setting the $P(i)^{th}$ to the $P(i+1)^{th}$ columns to 1, and filling the remainder of the rows with 0's. 6. The pseudoinverse of $\mathbf{B}$, $\mathbf{B}^+$ is computed. 7. $\mathbf{B}^+$ is postmultiplied by the position error matrix $\mathbf{e}$, to determine the interval error $\delta_{P(i-1),P(i)}$, where $0 \leq i < L-1$.

2.4 Iterative Solution of the Error Equations

A straight computation of the pseudoinverse-based solution is very memory intensive when L, and consequently N_s, are large. To facilitate the implementation of a stand-alone compensation algorithm, a version of the learning algorithm that requires minimum memory storage is sought. A simple stochastic gradient descent method is chosen, for implementation on a sample-by-sample basis. This is one

of the best known and oldest non-stationary methods. In spite of having large sequence lengths and slower convergent rate, the gradient descent method needs much less memory, as only a small number of vectors must be stored [70]. It is possible (and simpler) to implement a similar algorithm based on (2.3) in which only $\delta_{P(i)}$ and $\delta_{P(i-1)}$ are adjusted in any iteration of the learning algorithm, but convergence to correct line positions is slower and less smooth than with the proposed algorithm for velocities of greater than one code change per sample interval.

The CSDT velocity output at sample i of the velocity control system gives an estimate for the sum of the line width errors from the encoder lines $P(i-1)$ to $P(i)$. An initially zero vector is set up to represent the vector of slit width errors. Therefore, it is required to update the interval error values in a convergent manner. As described in (2.14), $e(i)$ is an updated estimate for $\sum_{k=P(i-1)}^{P(i)-1} \delta_{k,(k+1)}$. When the velocity is positive and the sensor output does not pass through the zero marker during an interval, so that $P(i) > P(i-1)$,

$$\begin{aligned} \delta_{k,(k+1)}(i) &= \delta_{k,(k+1)}(i-1) + \frac{e(i) - \sum_{y=P(i-1)}^{P(i)-1} \delta_{y,(y+1)}}{M(i)} \cdot \alpha(i), \\ & \forall k \in [P(i-1), P(i)-1],\ i \in [1, N_s] \end{aligned} \tag{2.23}$$

where $M(i) = P(i) - P(i-1)$, $\alpha(i)$ is a variable gain term and y is an index variable. Effectively, the residual error in slit widths, $e(i)$, is assumed to be equally divided between all relevant line intervals. As the sum of all slit/interval errors should be zero over the circumference of the disk, $-e(i)$ can be distributed among the

remaining intervals using

$$\delta_{k,(k+1)}(i) = \delta_{k,(k+1)}(i-1) + \frac{e(i) - \sum_{y=P(i-1)}^{P(i)-1} \delta_{y,(y+1)}}{L - M(i)} \cdot \alpha(i),$$
$$\forall k \notin [P(i-1), P(i)-1],\ i \in [1, N_s], \tag{2.24}$$

though this has proved unnecessary in practice when $M(i)$ is much smaller than L, because of the inconsequential influence of an application of (2.24) on each interval width, particularly when the update gain, $\alpha(i)$, decreases with time, as described below. Obvious modifications are made to (2.23) and (2.24) when the digital position value rolls through zero during the interval, where $P(i) < P(i-1)$, so that,

$$\delta_{k,(k+1)}(i) = \delta_{k,(k+1)}(i-1) + \frac{e(i) - \sum_{y=P(i-1)}^{L-1} \delta_{y,(y+1)} - \sum_{y=0}^{P(i)-1} \delta_{y,(y+1)}}{M(i)} \cdot \alpha(i),$$
$$\forall k \in [P(i-1), P(i)-1],\ i \in [1, N_s] \tag{2.25}$$

and,

$$\delta_{k,(k+1)}(i) = \delta_{k,(k+1)}(i-1) + \frac{e(i) - \sum_{y=P(i-1)}^{L-1} \delta_{y,(y+1)} - \sum_{y=0}^{P(i)-1} \delta_{y,(y+1)}}{L - M(i)} \cdot \alpha(i),$$
$$\forall k \notin [P(i-1), P(i)-1],\ i \in [1, N_s] \tag{2.26}$$

respectively. Similarly when the velocity is negative and the sensor output does not pass through the zero marker, where $P(i) < P(i-1)$, (2.23) and (2.24) are

re-written respectively as,

$$\begin{aligned}\delta_{k,(k+1)}(i) &= \delta_{k,(k+1)}(i-1) + \frac{e(i) - \sum_{y=P(i)-1}^{P(i-1)} \delta_{y,(y+1)}}{M(i)} \cdot \alpha(i), \\ &\forall k \in [P(i)-1, P(i-1)],\ i \in [1, N_s] \end{aligned} \tag{2.27}$$

and,

$$\begin{aligned}\delta_{k,(k+1)}(i) &= \delta_{k,(k+1)}(i-1) + \frac{e(i) - \sum_{y=P(i)-1}^{P(i-1)} \delta_{y,(y+1)}}{L - M(i)} \cdot \alpha(i), \\ &\forall k \notin [P(i)-1, P(i-1)],\ i \in [1, N_s], \end{aligned} \tag{2.28}$$

When the sensor output passes though the zero marker with negative velocity, so that $P(i) > P(i-1)$, (2.25) and (2.26) are modified as,

$$\begin{aligned}\delta_{k,(k+1)}(i) &= \delta_{k,(k+1)}(i-1) + \frac{e(i) - \sum_{y=P(i)-1}^{L-1} \delta_{y,(y+1)} - \sum_{y=0}^{P(i-1)} \delta_{y,(y+1)}}{M(i)} \cdot \alpha(i), \\ &\forall k \in [P(i)-1, P(i-1)],\ i \in [1, N_s] \end{aligned} \tag{2.29}$$

and,

$$\begin{aligned}\delta_{k,(k+1)}(i) &= \delta_{k,(k+1)}(i-1) + \frac{e(i) - \sum_{y=P(i)-1}^{L-1} \delta_{y,(y+1)} - \sum_{y=0}^{P(i-1)} \delta_{y,(y+1)}}{L - M(i)} \cdot \alpha(i), \\ &\forall k \notin [P(i)-1, P(i-1)],\ i \in [1, N_s] \end{aligned} \tag{2.30}$$

respectively.

The choice of gain term $\alpha(i)$ was based on the fact that the actual slit width errors, $\delta_{k,(k+1)}$, do not change dynamically during the learning phase. Therefore, the compensation table seeks to determine the average of each line error, so that $\alpha(i)$ should be inversely proportional to the number of estimates made, based on particular $P(i)$ or $P(i-1)$ positions. The simple update formula,

$$\alpha(i) = \frac{1}{\lceil \frac{i}{L} \rceil} \tag{2.31}$$

where $\lceil x \rceil$ is the smallest integer that exceeds x, has been found to result in fast convergence, while minimizing the effect of a corrupted data point at the end of the data set. Given that the error e is nominally white in nature, and assuming that the reference velocity has little error, the relevant variances are given by $\text{var}(\mathbf{e}) < 2\ \text{var}(\mathbf{\Delta_A})$, the precise ratio depending on the correlation between $\delta_{P(i)}$ and $\delta_{P(i-1)}$. A further safeguard against outliers can be implemented by omitting data for which $|e(i)|$ is found to be excessive. Assuming positive rotation, Table 2.3, below, details the sequential learning algorithm. For negative rotation, (2.23) to (2.26) are replaced by (2.27) to (2.30).

2.5 Conclusions

This chapter presents three different learning algorithms: Pseudoinverse-A, Pseudoinverse-B and a simple iterative method. The proposed Pseudoinverse-based algorithms are based on the least-mean-squares concept to calculate slit error, δ_i, and interval error, $\delta_{i,i+1}$, respectively. Pseudoinverse-based methods might need

TABLE 2.3: Sequence of operations performed to implement Iterative learning algorithm, assuming positive shaft velocity

Iterative learning algorithm
Initialization : $\delta_i = 0$; where i is an integer in the range $0 \leq i < L$.
1. N_s number of encoder events, $P(i)$ and $T_a(i)$, are obtained from the FPGA. 2. The average velocity over the interval, $V(i)$, is calculated using the CSDT method. 3. The reference velocity, $V_r(i)$, is estimated by various techniques like curve-fitting, or by filtering the high-frequency components from $V(i)$. 4. Using $V_r(i)$ and $V(i)$, the differential line error, $e(i)$, is calculated. 5. This differential line error, $e(i)$, is proportionally divided among the positions between $P(i-1)$ and $P(i)-1$ using (2.23) or when the shaft passes through the ZM during a cycle, (2.25) is used. 6. As the summation of interval errors should be zero, $-e(i)$ is proportionally distributed, using (2.24) or when the shaft passes through the ZM using (2.26), among the remaining encoder positions. 7. Repeating the process from Step 2 to Step 6, for all N_s samples, will lead to the convergence of all interval errors of the encoder. 8. Using the calculated interval errors, slit errors of the encoder can be determined, using $\delta_0 = 0$.

more memory, but the computation speed is faster than an iterative method. Such an iterative method inspired by the stochastic gradient descent method, for on-line implementation, was also proposed to determine the interval errors of an encoder. It was observed in several studies that the pseudoinverse methods are capable of performing very accurate calculation of the slit error with far fewer samples than the iterative one. However, for on-line implementation, where availability of memory storage is limited, the iterative method is a viable option.

Chapter 3

Error Modeling and Analysis

3.1 Introduction

The traditionally-quoted sources of error in the encoder were described in Section 1.2. In Chapter 2, where the learning algorithm was proposed under the assumption that a reference velocity signal could be estimated, it was assumed that the error in transition location, $\delta_{P(i)}$, was a fixed quantity. A more detailed description of the possible sources of error in the CSDT, and the manner in which these errors can be included in a model of the encoder/CSDT, is presented in this Chapter. As with any model of a system with a multi-factorial set of error sources, the model will give a realistic and typical, as opposed to definitive, indication of the exact influence of each error source.

When a very high-inertia wheel is connected rigidly to the sensor, one can expect that the actual motor velocity will vary smoothly, so that the reference velocity,

V_r, used in the learning algorithm will approximate very closely to the actual shaft velocity. However, even in the case that the drive inverter is unpowered (during a run-down test) a small residual difference is likely between the reference velocity and a compensated CSDT output velocity estimate. This is analyzed further below. It is worth noting that the errors in the compensated CSDT velocity are very small, so that second-order effects which would not be detected in many velocity estimation systems, become significant. Even the small changes in friction in the mechanical system as the motor rotates, due for example to imperfect bearings, may have an effect.

In this chapter, attention is not paid to which of the learning algorithms described in the previous chapter is used to implement the compensation. The pseudoinverse-based algorithms require more memory for implementation, but the computation time is short. The results and discussion on the various learning algorithms is delayed until Chapter 4 when results for real encoder data informs the comparison and discussion.

3.2 Error Modeling with Random Noise Included

The accuracy of the average shaft velocity, V, calculated over a sample period using the CSDT method, is affected by errors from various sources. A realistic mathematical model of the CSDT velocity error is developed for analyzing the effects of these errors. In this analysis, slit error, $\delta_{P(i)}$, or interval error, $\delta_{i-1,i}$, are assumed to occur due to manufacturing non-idealities in the code-wheel, while

the low-frequency error ($\delta^{lf}_{P(i)}$) is assumed to be due to code-wheel eccentricity, radial play, and mis-alignment between shaft and encoder. Additionally, random noise errors, $\delta^{n}_{P(i)}$ (related to position $P(i)$), and $\delta^{n}(i)$, which are independent of position, are assumed to occur due to quantization of the auxiliary timing circuitry and variations in the shaft velocity. However, the quantization error, which is commonplace in velocity measurement, is relatively very small due to the time-stamping employed in the CSDT method. As the actual velocity, $V_{act}(i)$, of the shaft is not known, the reference velocity, $V_r(i)$, which can be generated by using several techniques, as described in Section 3.4, will not equal the actual shaft velocity, so a reference velocity error, $\delta^{r}_{P(i)}$, must be added to any model of the system. An apparent shaft velocity, V_I, calculated at the i^{th} sampling instant using the CSDT method, can be written as,

$$V_I(i) = \frac{P(i) - P(i-1)}{T(i)}, \tag{3.1}$$

where $P(i)$ and $P(i-1)$ are the digital positions of the encoder at the i^{th} and $(i-1)^{th}$ sampling instants and $T(i)$ is the time taken by the shaft to rotate from digital position $P(i-1)$ to $P(i)$. This gives the correct velocity only in an ideal system with perfect transition locations. The above equation can also be expressed in terms of interval counts or pulse counts, as

$$V_I(i) = \frac{M(i)}{T(i)}, \tag{3.2}$$

where $M(i)$ is the number of pulse counts over time interval $T(i)$. The error

modeling can be performed by considering either slit error, $\delta_{P(i)}$, or interval error, $\delta_{i-1,i}$. In the initial case study, the analysis will be performed by considering slit errors and, later, similar studies where performed for interval errors, but these are not included in the thesis, as they did not show any further insite into the operation of the encoder compensation.

3.2.1 Error modeling considering slit error, $\delta_{P(i)}$

If the error in this estimate were due entirely to transition location error ($\delta_{P(i-1)}$ and $\delta_{P(i)}$ are slit errors at the $P(i-1)^{th}$ and $P(i)^{th}$ edge positions, respectively), the actual average velocity over the measurement interval would be

$$V_{act}(i) = \frac{[P(i) + \delta_{P(i)}] - [P(i-1) + \delta_{P(i-1)}]}{T(i)} \tag{3.3}$$

There will be electrical noise associated with the photo detector outputs, and the combination of the analogue filtering and the hysteresis built into the comparator circuitry will lead to a change in the actual transition location seen at a particular nominal sample. This will occur to some extent despite the probable use of a scanning reticule with multiple light paths, and of the associated analogue and/or digital filtering intended to minimize the noise. While the data used in the learning algorithm is effectively averaged over many thousand samples, so that the aforementioned noise need not be considered when analyzing the learning algorithm, no such averaging is present when computing figures of merit for the

CSDT output, compensated or otherwise, as in that case it is a single CSDT output and a single velocity reference estimate that are used. Similarly, when both slit error and additional random noise, $\delta^n_{P(i-1)}$ and $\delta^n_{P(i)}$ (random noise induced at $P(i-1)^{th}$ and $P(i)^{th}$ edge positions respectively) are considered, shaft velocity can be approximated by

$$V_{act}(i) = \frac{[P(i) + \delta_{P(i)} + \delta^n_{P(i)} + \delta^n(i)] - [P(i-1) + \delta_{P(i-1)} + \delta^n_{P(i-1)} + \delta^n(i-1)]}{T(i)} \tag{3.4}$$

where $\delta^n(i)$ is a random error which is not dependent on $P(i)$, while $\delta^n_{P(i)}$ is that portion of the random or variable error that varies with $P(i)$. It is assumed that the three proposed learning methodologies provide a very good estimate of the fixed error $\delta_{P(i)}$, in the shaft velocity, so that the $\delta_{P(i)}$ should converge to the correct value despite the presence of errors in the reference velocity values. The simulation of the learning algorithms verifies the fact that the estimates of any constant errors in the transition locations converge to zero as the number of samples taken increases. However, in practice, some high-frequency components are found in the compensated velocity output, and this random noise will create an upper limit on the improvement that can be obtained by a compensation method when the system is operating in a noisy environment; i.e. some sources of error remain. The resultant compensated shaft velocity, $V_c(i)$, obtained from these methods, can be expressed as

$$V_c(i) = \frac{[P(i) + \acute{\delta}_{P(i)}] - [P(i-1) + \acute{\delta}_{P(i-1)}]}{T(i)} \tag{3.5}$$

where $\acute{\delta}_{P(i)}$ is the transition error estimate obtained from the learning algorithm. The corresponding nett error in the transition location used in the CSDT is

$$\delta_{P(i)} + \delta^n_{P(i)} + \delta^n(i) - \acute{\delta}_{P(i)}$$

A good compensation algorithm will therefore eliminate $\delta_{P(i)}$, and leave the other residual sources of velocity error estimation unaltered. This scenario will be included in a simulation platform for cross-checking the effectiveness of the proposed algorithm. Using the above error assumptions

$$T(i) \cdot V_{err} = [\delta_{P(i)} + \delta^n_{P(i)} + \delta^n(i)] - [\delta_{P(i-1)} + \delta^n_{P(i-1)} + \delta^n(i-1)] \tag{3.6}$$

before compensation. Similarly,

$$T(i) \cdot V_{err-c} = [\delta^n_{P(i)} + \acute{\delta}_{P(i)}] - [\delta^n_{P(i-1)} + \acute{\delta}_{P(i-1)}] \tag{3.7}$$

will be valid after the compensation.

Even if the effect of reference velocity error can be greatly reduced in the learning algorithm when estimating $\delta_{P(i)}$, it can still be a major component of the error obtained when estimating the error in any particular velocity estimate, either with or without compensation. The error is calculated using $\hat{V}_{err} = V_r(i) - V(i)$ before compensation, and $\hat{V}_{err-c} = V_r(i) - V_c(i)$ after compensation.

The actual root-mean-squared improvement in the CSDT performance due to compensation is given by the figure of merit, m,

$$m = \sqrt{\frac{\bar{V}_{err-c}^2}{\bar{V}_{err}^2}}, \tag{3.8}$$

where $\bar{V}_{err}$ and $\bar{V}_{err-c}$ are actual velocity errors in shaft velocity before and after compensation, respectively. Similarly, the calculated (apparent) improvement is obtained by

$$\acute{m} = \sqrt{\frac{\hat{V}_{err-c}^2}{\hat{V}_{err}^2}} \tag{3.9}$$

where $\hat{V}_{err}$ and $\hat{V}_{err-c}$ are the apparent velocity errors in shaft velocity before, and after, compensation, respectively.

Similar to the analysis performed when considering slit errors δ_i, an alternative and equivalent analysis can be performed by considering the interval error $\delta_{i-1,i}$. The average shaft velocity, V_{act}, in terms of number of pulse counts, $M(i)$, can be expressed as

$$V_{act}(i) = \frac{M(i) + \delta_{i-1,i} + \delta_{M(i)}^n + \delta^n(i)}{T(i)} \tag{3.10}$$

while $M(i)$ and $\delta_{i-1,i}$ are the number of encoder transitions that occur between the i^{th} and $(i-1)^{th}$ samples and interval errors. $\delta_{M(i)}^n$ and $\delta^n(i)$ are random noises which are dependent and independent of encoder position, respectively. The reference velocity, V_r, predicted using the zero-phase low-pass filter or a similar technique, can be expressed as

$$V_r(i) = \frac{M(i) + \delta_{i-1,i}^{act}}{T(i)} \tag{3.11}$$

where, $\delta^{act}_{i-1,i}$ is the position difference between actual and ideal positions. Similarly, shaft velocity after the compensation, V_c, can be expressed as,

$$V_c(i) = \frac{M(i) + \bar{\delta}_{i-1,i}}{T(i)} \tag{3.12}$$

where, $\bar{\delta}_{i-1,i}$ is the uncompensated component of the error in shaft velocity. The amount of error reduction, m, in the

$$m = \sqrt{\frac{(V_c(i) - V_r(i))^2}{(V_{act}(i) - V_r(i))^2}}. \tag{3.13}$$

A similar analysis was undertaken using interval errors, rather than slit errors. The resulting equations did not provide any additional insight into the errors associated with the encoder non-idealities, and so are not included in this thesis.

3.3 Effect of sinusoidally distributed error on shaft velocity

The lack of an ultra-high-accuracy reference tachometer and the probability of a small low-frequency (once-per-revolution) variation in the shaft velocity due to gravitational effects, lead to a fundamental problem: one cannot distinguish between a small steady-state velocity variation that is correlated to the shaft position and a variation in the transition location errors over the circumference

of the code-wheel which could give the same tachometer output in the absence of such a velocity variation.

The issue is illustrated by a small once-per revolution sinusoidal variation in the transition location errors for a motor rotating at a uniform speed. For simplicity, it is assumed that the transition locations are perfect except for this sinusoidal variation.

If the intervals between transition locations vary, the corresponding apparent CSDT velocity outputs will also vary. It is assumed that the transition location error varies over the circumference of the code-wheel, as follows:

$$\delta_{P(i)} = a_s \cdot \sin\left[\frac{2\pi P(i)}{L}\right] \tag{3.14}$$

where, a_s is a small constant, i.e. a simple sinusoidally distributed error. The corresponding CSDT output is,

$$V_I(i) = \frac{P(i) - P(i-1)}{T} \tag{3.15}$$

which gives the apparent velocity output, while the actual velocity is given by

$$\begin{aligned} V_{act}(i) &= \frac{(P(i) + \delta_{P(i)}) - (P(i-1) + \delta_{P(i-1)})}{T} \\ &= \frac{(P(i) + a_s \cdot \sin(\frac{2\pi P(i)}{L})) - (P(i-1) + a_s \cdot \sin(\frac{2\pi P(i-1)}{L}))}{T} \end{aligned} \tag{3.16}$$

From the above equation, the ratio of CSDT velocity V to actual velocity V_{act} can be calculated as

$$\frac{V_I}{V_{act}} = \frac{1}{1 + \frac{a_s[\sin(\frac{2\pi P(i)}{L}) - \sin(\frac{2\pi P(i-1)}{L})]}{P(i) - P(i-1)}}. \tag{3.17}$$

If it is assumed that V_{act} is constant, and if it is desired to consider the apparent velocity output, in the limit, when the position change per sample interval is a very small part of a full rotation,

$$\begin{aligned}
&\frac{\sin(\frac{2\pi P(i)}{L}) - \sin(\frac{2\pi P(i-1)}{L})}{P(i) - P(i-1)} \\
\approx\ &\frac{2\cos(\frac{2\pi}{L}\frac{P(i)+P(i-1)}{2})\sin(\frac{2\pi}{L}\frac{P(i)-P(i-1)}{2})}{P(i) - P(i-1)} \\
\approx\ &\frac{2\pi}{L}\cos(\frac{2\pi}{L}P(i))\frac{\sin(\frac{\pi}{L}(P(i) - P(i-1)))}{\frac{\pi}{L}(P(i) - P(i-1))} \\
\approx\ &\frac{2\pi}{L}\cos(\frac{2\pi}{L}P(i))
\end{aligned} \tag{3.18}$$

as $\sin(x)/x \approx 1$ for small x. From the above, equation (3.17) can be re-written as,

$$\begin{aligned}
\frac{V_I}{V_{act}} &\approx \frac{1}{1 + a_s\frac{2\pi}{L}\cos(\frac{2\pi}{L}P(i))} \\
&\approx 1 - a_s\frac{2\pi}{L}\cos(\frac{2\pi}{L}P(i))\,,
\end{aligned} \tag{3.19}$$

the mean CSDT velocity output being $V_{mean} = V_{act}$. If the encoder were perfect, and the actual shaft velocity was given by

$$V_{act} = V_{mean}(1 - a_s \frac{2\pi}{L} \cos(\frac{2\pi}{L} P(i))), \tag{3.20}$$

the CSDT output would match that of the previous case with the imperfect encoder.

This demonstrates that apparent velocity changes due to low-frequency code-wheel errors or code-wheel non-orthogonality can appear as correlated velocity changes over the shaft revolution. These effects are very small. Equivalently, as previously mentioned, the velocity error due to the differentiation of small low-frequency position errors will be small[1], something easily verified with a simulation model.

3.4 Simulation of Encoder, Tachometer and Iterative Learning

A MATLAB-based simulation is used to implement the algorithms. For analysing the effects of these encoder errors, a quasi-real-time incremental encoder with a resolution of L ppr, consisting of low-frequency error (which can be due to the

[1]It is fortuitous that the encoder non-ideality which cannot be distinguished from a once-per-revolution velocity variation is very small. Therefore, such once-per-revolution apparent velocity changes are filtered out in the learning algorithm. If one wished to distinguish the effects of correlated velocity and code-wheel errors over the mechanical cycle, a multi-test arrangement, in which the encoder shaft was rotated relative to the motor shaft between tests, would allow measurement of the real low-frequency transition error variation to be isolated. However, as the primary focus of the work being reported relies on the ability to perform in-situ learning, such a method is not attractive.

inaccuracies like interval error, axial play, and eccentricity), and high-frequency error (caused by slit error and random noise - where some random error is position dependent and some is position independent), was simulated. At each sample instant, a search-type algorithm is utilized to determine the last encoder transition encountered, and consequently the appropriate pulse-count and quantized auxiliary time data obtained at each sample instant for any given shaft velocity profile. To demonstrate its ability to perform learning at variable speeds, data obtained from a simulated ramp-down velocity profile is given as input to the simulation model. The iterative learning algorithm is implemented as explained in Table. 2.3. The generation of V_r is intrinsic to the operation of all the learning algorithms, because the difference between the CSDT-derived velocity and V_r allows the estimation of the encoder line errors. This is performed using MATLAB **lsqcurvefit**, **butter** & **filtfilt** functions. The non-linear curve-fitting function, **lsqcurvefit**, using least-square solutions, performs a curvefit for the CSDT velocity when the profile is known to be predictable. By utilizing the **butter** and **filtfilt** functions, a zero-phase filtering of the CSDT velocity array can be performed to obtain the reference velocity array. In addition to this filtering method, other techniques such as: averaging the velocity over the previous few samples, or extrapolation techniques, can also be utilized to calculate V_r (as will be detailed in the next chapter). Using the calculated reference velocity, V_r, the inaccuracies, $\delta_{k,(k+1)}$, are calculated at each sample. Implementing (2.23) and (2.24) during an interval without a zero crossing, i.e. when the zero-marker is not active, the error is distributed among all intervals, as explained in Section 2.4. Similarly, (2.25) and (2.26) are implemented when a zero crossing occurs. Repeating (2.23)-(2.26) over N_s samples is found to

allow all the interval errors to converge towards their true values. The interval errors can be obtained and then used to recalculate the actual slit positions.

Because the actual transition positions of the simulated encoder are known, it is possible to calculate the exact distance between two transition positions. The nature of the simulated encoder errors was explained in Section 2.2. The output of this simulated encoder can be calculated for a required velocity profile. (Obviously, a prior knowledge of the actual encoder position errors is not assumed in the simulation of the CSDT, to match the reality that no precise measurement equipment is used). The calculated position error is used to compute the slit and interval errors using (2.23) and (2.24), as proposed previously. The calculated (estimated) slit errors are compared to the (pre-chosen) slit errors of the simulated encoder. It is clear from Fig. 3.1 that the learning algorithm accurately determines the slit errors corresponding to the non-ideal encoder transition positions $\bar{P}_i$

$$\bar{P}_i = P_i + \delta_i + a_s \cdot \sin\left[\frac{2\pi P(i)}{L}\right] + \delta^n_{P(i)} \tag{3.21}$$

consisting of both low and high frequencies, as explained earlier in this chapter.

The performance improvement of the CSDT velocity error (above 90 %), when taking advantage of the revised encoder transition positions, using the iterative learning method, is seen in Fig. 3.2, which shows the simulated estimated and actual shaft velocities before, and after, compensation.

The effect of the position-independent, random noise error, $\delta^n(i)$, component can be analyzed by adding the error values to the auxiliary time of a CSDT sampling

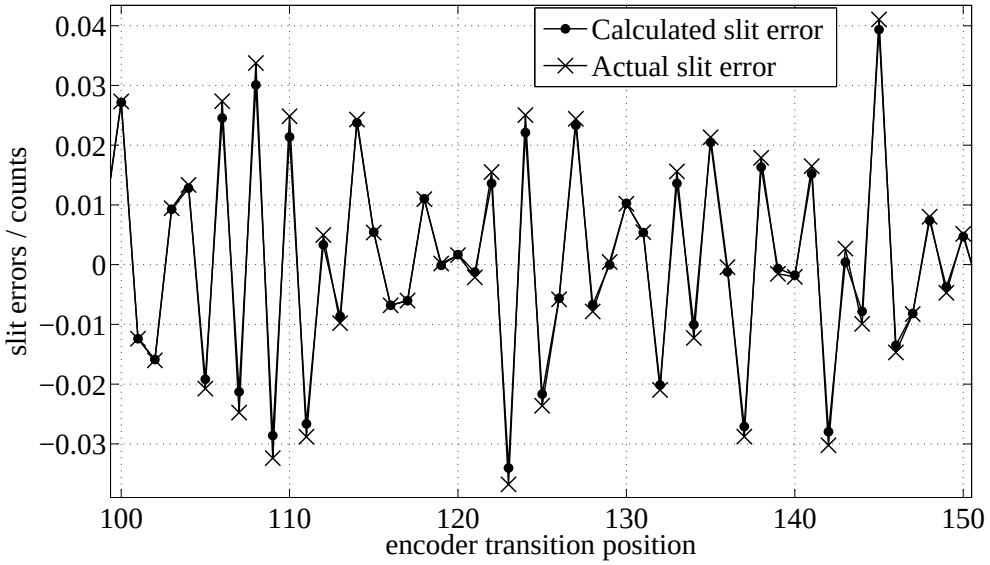


FIGURE 3.1: Actual and calculated slit errors for nominal encoder transition positions of 100 to 150, with the learning algorithm incorporated into the simulation program.

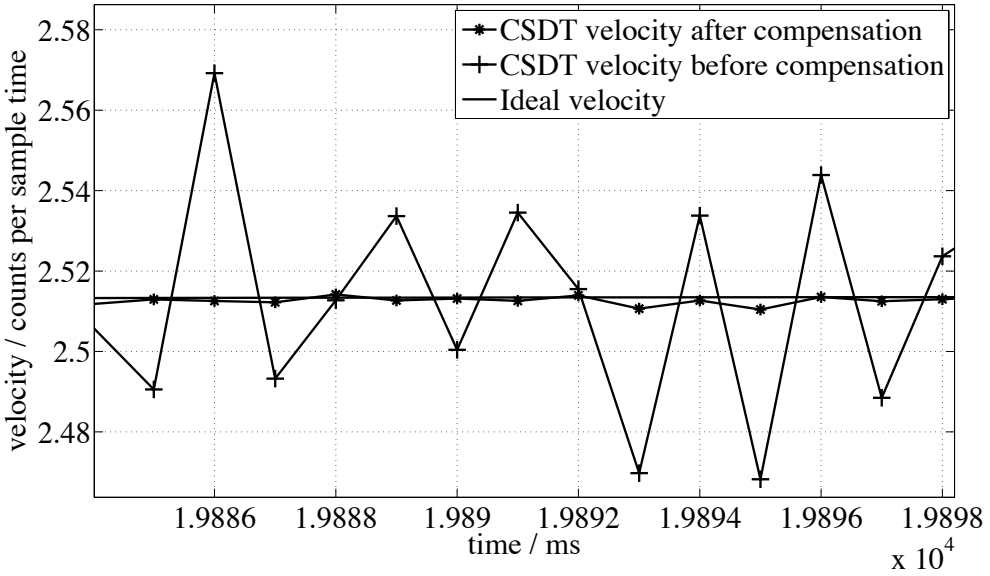


FIGURE 3.2: Simulation outputs comparing actual shaft velocity with CSDT velocities, before and after compensation, (for $T_s = 1$ ms).

event. Fig. 3.3 shows the performance of the iterative learning algorithm versus different quantities of random noise at the sampling instants. It can be seen that the performance (figure of merit, m) of the iterative learning algorithm decreases almost linearly with an increase in the mean rms value of random noise per encoder position. In the absence of a random error, it can be observed from Fig. 3.3 that the algorithm almost eliminates the effect of $\delta_{P(i)}$, with a figure of merit, m, of approximately 99.43%.

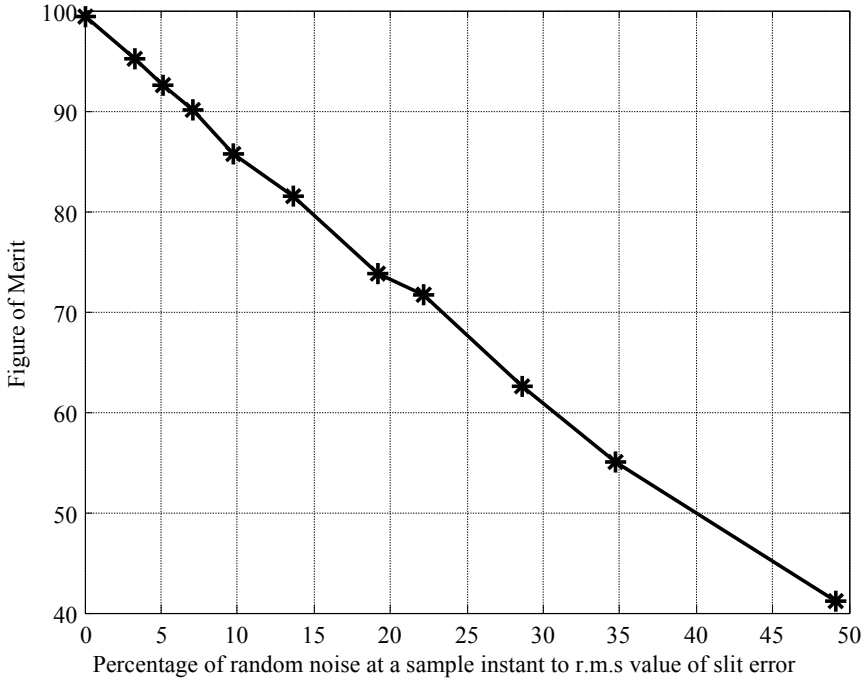


FIGURE 3.3: Performance of iterative algorithm for varied amounts of random noise, using simulated encoder data

Similarly, the performance of the iterative learning algorithm is analyzed by varying the magnitude of the low-frequency velocity error. The results obtained are plotted in Fig. 3.4, where it can be seen that the figure of merit does not vary significantly as the magnitude of the added low-noise is changed. Therefore, it can be surmised that low-frequency transition location is effectively removed by a

zero-phase Butterworth filter with a bandwidth of 100 Hz, thereby minimizing the effect of this error source on the performance of the learning algorithm. Equivalently, it is clear that the low-frequency position error does not have a large or systematically varying effect on tachometer error.

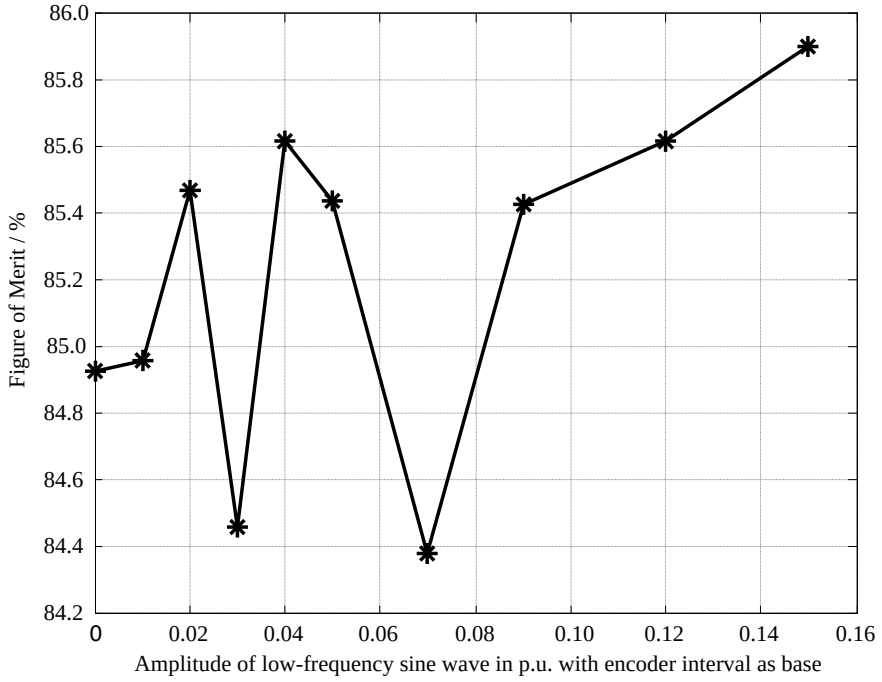


FIGURE 3.4: Performance of iterative algorithm with different proportions of added low-frequency noise, using real encoder data

Little extra insight is gained by simulating the pseudoinverse-based learning algorithms. Instead, the comparisons are performed experimentally on physical systems. These are described in Chapter 4.

3.4.1 Investigation of various means of calculating the reference velocity

In order to minimize the cost of the actual implementation of the system, no additional reference encoder or equipment are used. Therefore, the generation of a reference velocity that closely matches the actual shaft velocity, is of vital importance. Some of the methods proposed to obtain the reference velocity are: use of a zero-phase filter, averaging of previous samples, and interpolation techniques.

1. **Zero-phase filter** : Use of a zero-phase filter is a good method for generating the reference velocity, V_r, where the zero-phase filter will perform filtering in both forward and reverse directions. Using this technique, the output filter signal will not see any lag or lead due to filtering, so that zero-phase distortion is observed in the output signal. However, the order of the zero-phase filter will be twice the order of the specified filter type. It is important to state that the zero-phase filter cannot operate in real-time, but that there is little difficulty in applying the filter to stored data. When implementing this in MATLAB, the **filtfilt** function can be used. For experimental implementation, the CSDT velocity data need to be filtered once in the forward direction and once in the reverse direction.

2. **Mean velocity** : This is the simplest of the three methods considered, where the reference velocity at the i^{th} sample, $V_r(i)$, is the mean of the last n samples. This technique can be implemented on any kind of velocity profile, and the time lag will not be long if n is not large, so that it will

often be suitable for real-time implementation of the learning algorithm. The expression to calculate the reference velocity is simply

$$V_r(i) = \frac{V(i) + ... + V(i-n+1)}{n} \tag{3.22}$$

3. **Interpolation** : Another possibility is to use a variant of the polynomial-fit method of Merry *et al.* [4, 5], whereby interpolation is performed on a second-order polynomial fitted to a number of time-stamped data points, as exemplified by the point $(i\,T_s - T_a(i),\, P(i)\,)$. For off-line applications, the fitted data points correspond to times that are both before and after sample i, and the resulting velocity estimate at the centre of the measurement interval $T(i)$ is used as a reference estimate. Similar to the technique based on the calculation of mean velocity, it can also be used in real-time implementation of learning algorithms when only previous data points are used.

3.4.2 Behaviour of slit and interval errors

As a first-order approximation for the purpose of simulation, the slit errors can be considered independent and random, and to be uniformly distributed. The interval error of the encoder usually has a low frequency, approximately sinusoidal, component, with a fundamental frequency of one per revolution [19]. These errors, as in Figs 3.5 and 3.6, for a simulated encoder with 100 counts per revolution, can be produced by the addition of a sinusoidal component and random noise in a

MATLAB-based simulation. Additionally, other errors can also be included, as previously discussed.

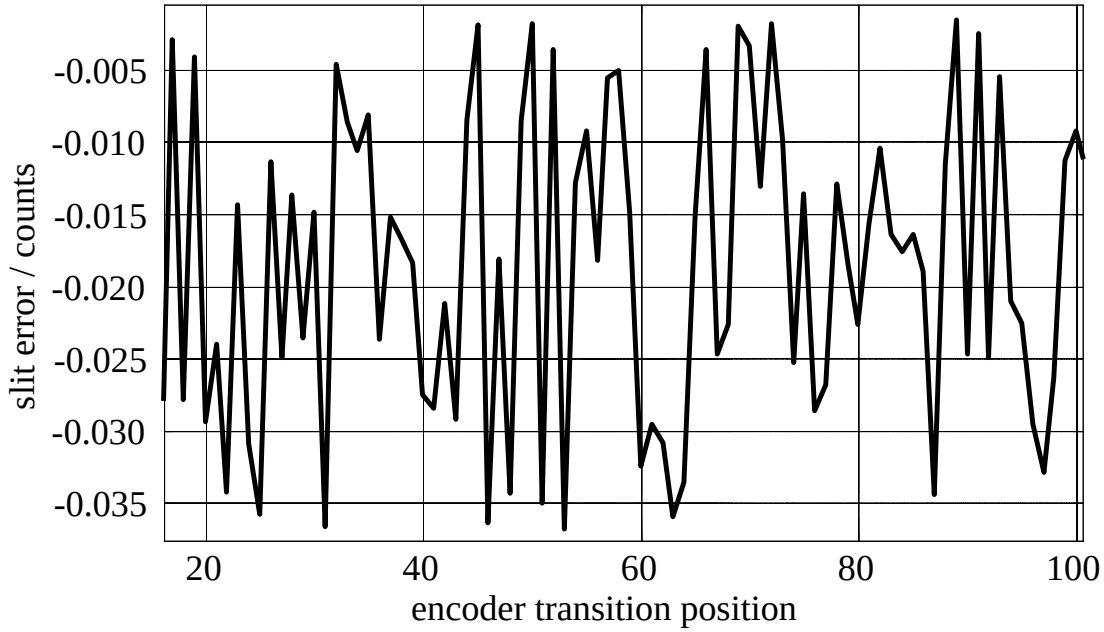


FIGURE 3.5: Slit error profiles generated from a MATLAB-simulated incremental encoder

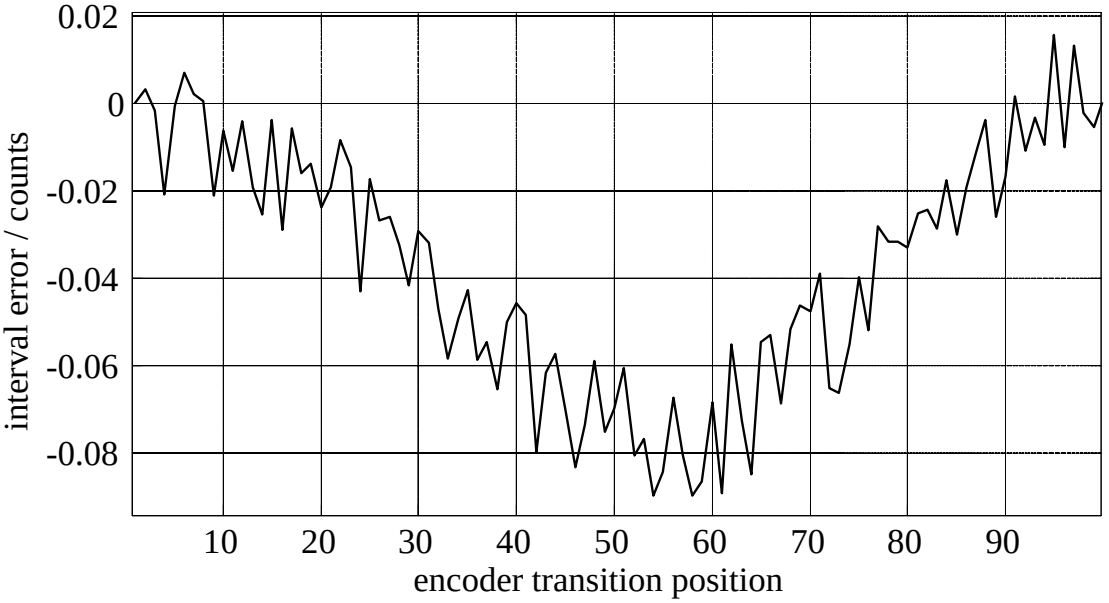


FIGURE 3.6: Interval error profile generated by a MATLAB-simulated incremental encoder

As has been analysed by Kavanagh in [22], a uniform distribution of encoder transition position errors can lead to a triangularly distributed differential (velocity) errors, depending on the correlation between nearby encoder transition errors. It is the differential error that is significant in terms of the velocity errors inherent in a tachometer. The probability density function (p.d.f.) relating to the differential errors associated with a non-ideal simulated encoder has been found to be closer to that due to a truncated Gaussian distribution, than to a uniform, distribution, as described by Kavanagh in [71]. Both models predict very similar velocity error characteristics. Therefore, the simpler uniform distribution can be employed for tachometer analysis. Fig. 3.7 shows that the proposition that the interval error and slit error have approximately triangularly and uniformly distributed natures, respectively, is reasonable.

It should be noted that the apparently skewed error distribution associated with the slit error is due to the assumption in the simulation that encoder line zero is defined to be at position zero, so that a once-per-revolution integral error around the circumference of the disk will usually lead to slit errors which are not centred about zero, on average. Specifically, the slit error is defined as zero at encoder transition position zero. When the array of errors is calculated using the appropriate formula and distribution, all values are shifted to ensure this zero value.

3.4.3 Use of real encoder data in tachometer simulation

The artificial encoder used in the simulation is replaced with real encoder interval errors (calculated using one of the learning algorithms) to analyze the variation

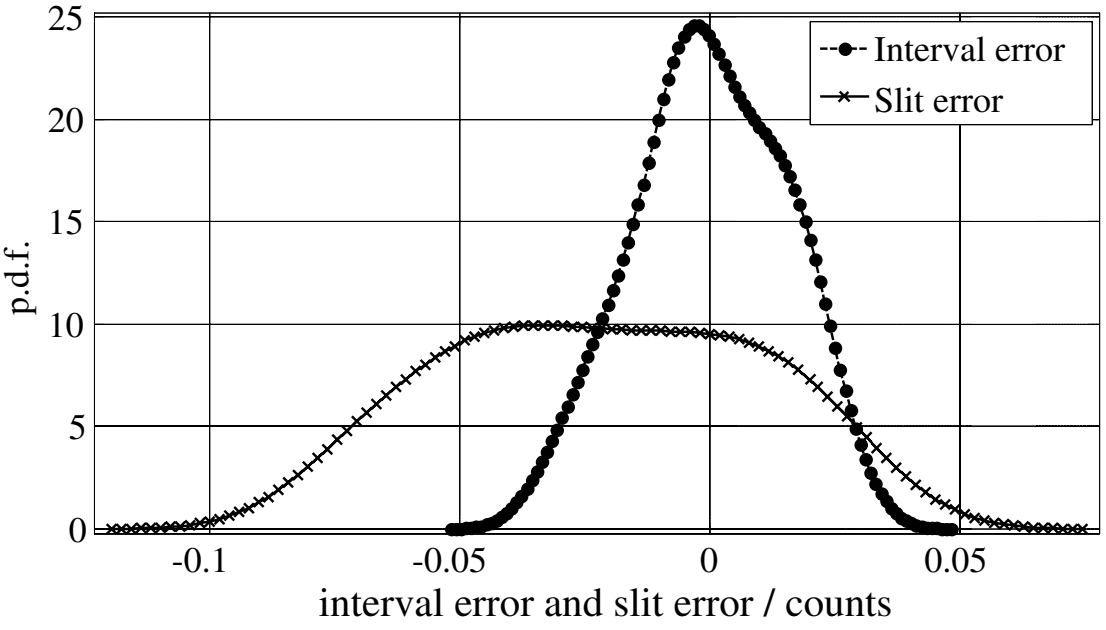


FIGURE 3.7: Probability density function of slit and interval errors generated from a MATLAB simulated incremental encoder

in the performance of learning algorithms due to random errors induced at the sampling instants. This performance analysis can be done by calculating the figure of merit, m, for different amplitudes of added random noise. Obtained results from this analysis are plotted in Fig. 3.8, where it can be observed that the learning algorithm can predict the interval/slit errors of the encoder almost perfectly when there is no random error. The ability of the algorithm to calculate the interval/slit error decreases with an increase in the magnitude of the random error, as might be expected.

For evaluating the influence of the position dependent($\delta^n_{P(i)}$) and independent ($\delta^n(i)$) noise on the effectiveness of the learning algorithm, simulations with different $\delta^n_{P(i)}$ and $\delta^n(i)$ values are performed. The results obtained, with three different r.m.s. position-dependent noise values, using a 360 ppr resolution, simulated encoder and the iterative algorithm are shown in Fig. 3.9. It can be seen from the

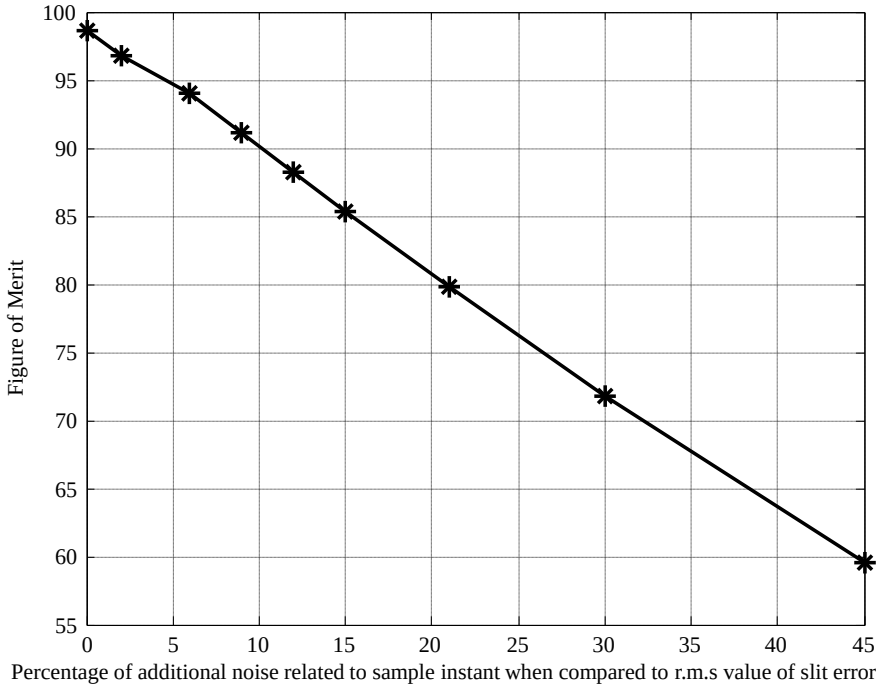


FIGURE 3.8: Performance of iterative algorithm with different proportions of added high-frequency noise

plot that the figure of merit, m, varies with the amount of position-dependent noise. Specifically, the ability of a learning algorithm, such as the iterative algorithm, to calculate the exact position of the encoder edge decreases when the non-position-dependent noise increases.

3.5 Conclusions

Three sources of a real or apparent CSDT error, even after compensation, have been identified as: random error due to electrical noise, errors in the reference velocity estimate used, and the inability of the algorithm to discriminate between small, repetitive physical velocity changes over the mechanical cycle and any alterations in the transition errors over the mechanical cycle that are correlated with

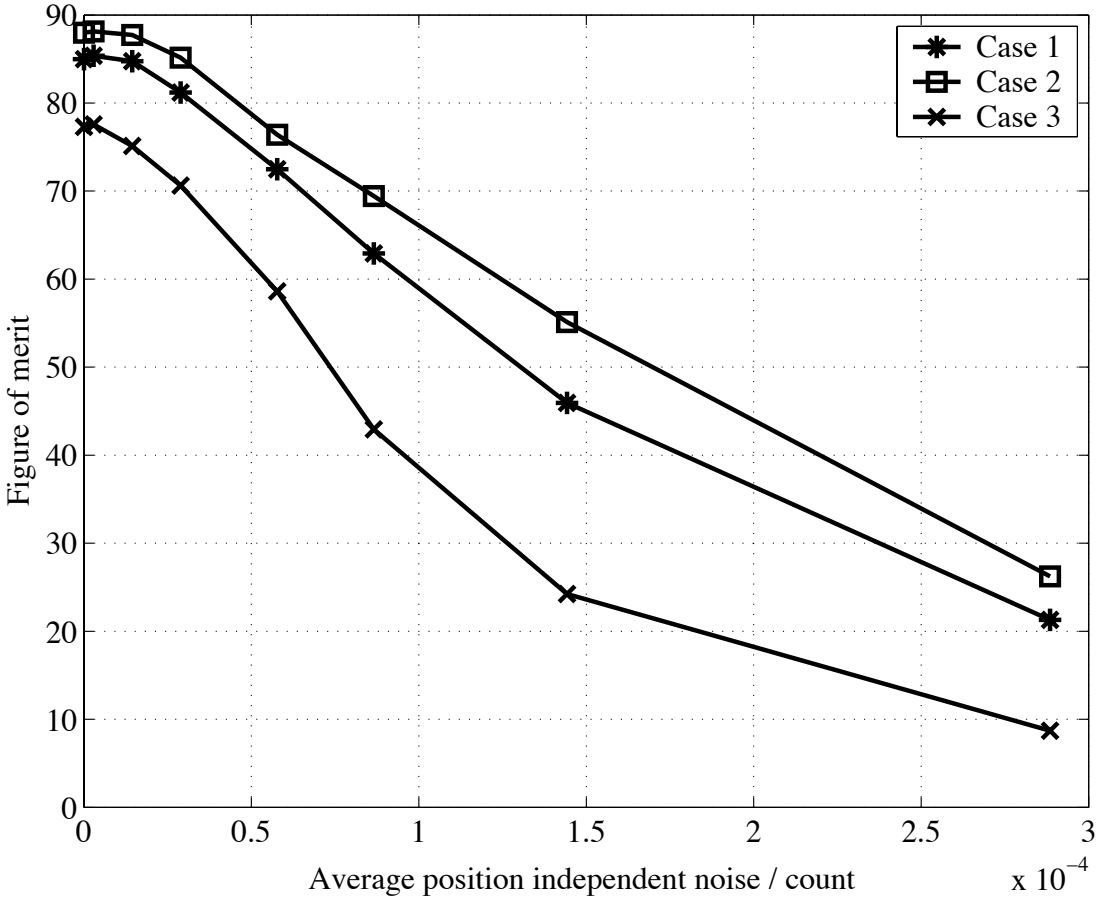


FIGURE 3.9: Comparing the effect of position independent error, $\delta^n(i)$, on the figure of merit for a simulated encoder of 360 ppr (where magniture of position-independent error is of the order Case 3 < Case 1 < Case 2).

the velocity changes. The latter error corresponds to once per-revolution velocity changes due to gravitational effects and changes to the encoder characteristics over the circumference (e.g. due to a non-orthogonal code-wheel or to a non-ideality in the code-wheel pattern at the fundamental frequency of the pattern). A theoretical analysis of these different error types was performed. Later in this chapter, the implementation of the learning algorithm on a simulation platform was presented. The results obtained from the simulations effectively support the theoretical analysis.

Chapter 4

Implementation and Evaluation of Learning Algorithms

4.1 Introduction

This chapter is dedicated to a description and analysis of the experimental implementation and analysis of proposed compensation techniques on incremental square-wave encoders. Initially, in Section. 4.2, the performance of learning techniques in an open-loop environment is analyzed. The FPGA implementation of the CSDT-based velocity measurement technique and an overview of the experimental hardware setup are discussed in Section. 4.2, and subsections 4.2.1 and 4.2.2 are dedicated to the presentation of the open-loop implementation of learning algorithms and a performance analysis of the results. In order to evaluate the consistency and repeatability of the proposed algorithms, several tests are performed

on incremental encoders of different quality and cost, with different resolutions, and from different manufacturers. In addition to testing the performance of the proposed learning algorithms, various filtering techniques to generate the reference velocity from the CSDT velocity are also implemented and compared for best performance. Finally, in Section. 4.3, the design of the learning algorithm in a closed-loop control application is presented.

4.2 Implementation and Experimental Evaluation of Learning Techniques in Open-Loop Measurement Applications

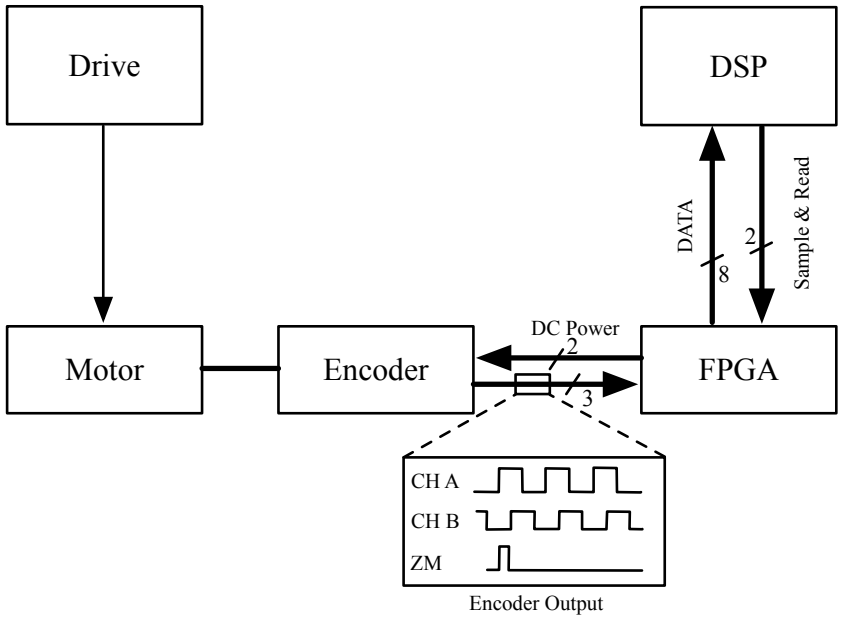


FIGURE 4.1: Block diagram of a typical FPGA/DSP-based servosystem controller.

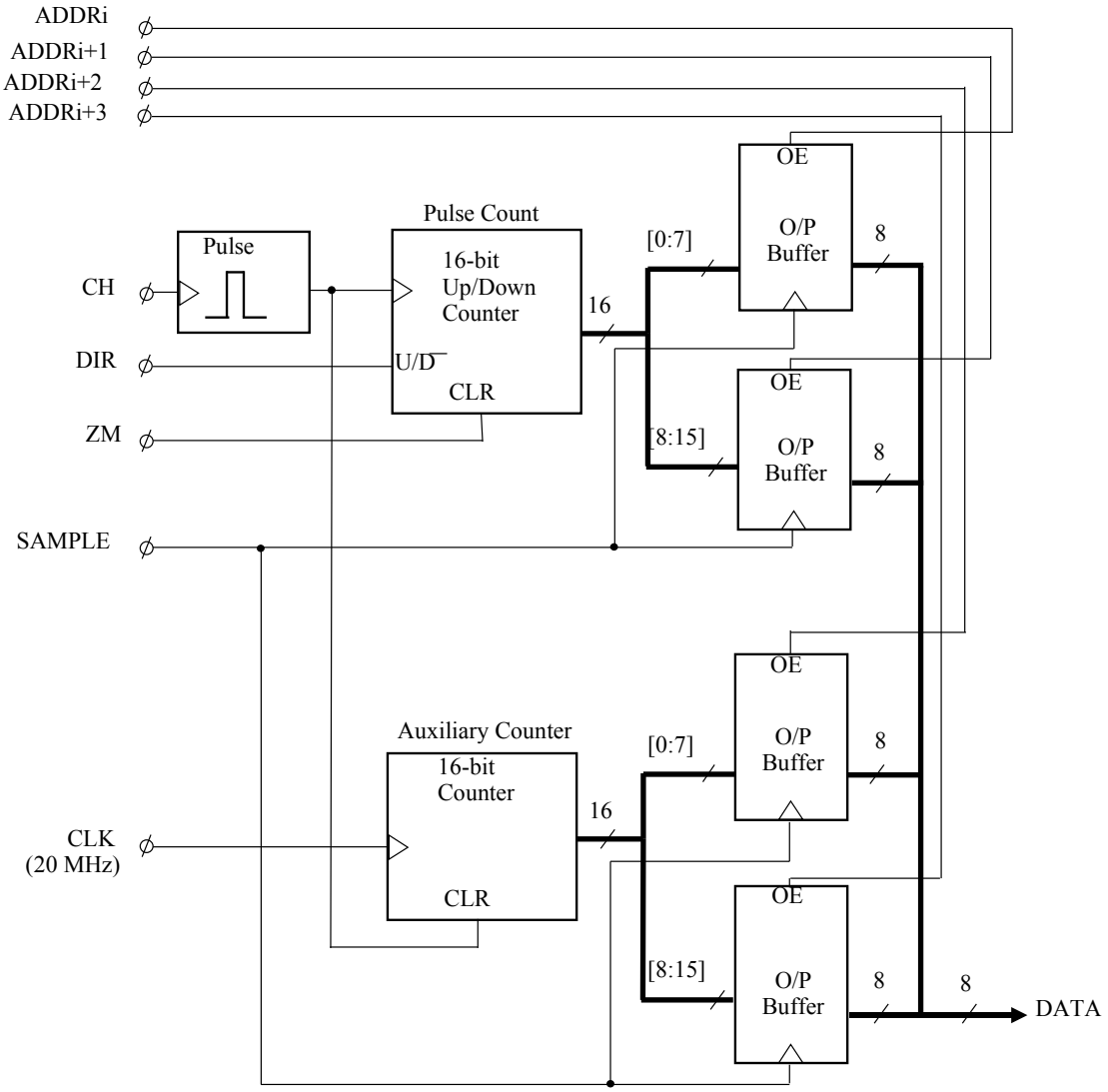


FIGURE 4.2: Block diagram of a FPGA/DSP-based CSDT implementation (reproduced for convenience).

The block diagram showing the experimental setup corresponding to an open-loop measurement implementation was depicted in Fig. 1.18. It is reproduced in Fig. 4.1, for convenience, where it can be seen that the pulse data from the outputs of three encoder channels are fed to a FPGA board that includes a basic Xilinx XC4025-series FPGA, operating with a clock speed of 20 MHz. Using this hardware, the necessary basic inputs to measure the CSDT velocity: digital position, $P(i)$, and auxiliary time, $T_a(i)$, can be calculated. Various logic circuits and clocks

are implemented on a Xilinx FPGA platform for these parameter calculations, as shown in Fig. 4.2, where a rising or falling edge of a channel A or B and a zero marker, ZM, can be used as raw input data to the FPGA. If necessary, the resolution of the velocity can be increased by a factor of two by considering both the rising and falling edges of a single A or B channel, and doubled again by considering the raising and falling edges of both channels A and B, which is termed quadrature decoding, as already described in Section 1.5.3. The implementation of quadrature decoding in the FPGA is performed in the BIPUL1 subsystem in Fig. 4.3, (where the output signals QUAD and QPLS DIR are the quadrature signal and its direction, respectively). The logic circuitry to calculate the pulse count and auxiliary time are designed in the CSDTBL_D block in Fig. 4.3. Filtering of the input encoder signals to this block is performed in INQPLUS.

Some of the other important signals shown in Figs. 4.2 and 4.3 are as follows:

ADDR	Address bus for selecting FPGA output register.
CH	Filtered output of shaft encoder channel, where the signal can contain positive and/or negative edges of CH-A and/or CH-B.
DATA	8-bit FPGA data bus, sent to DSP.
DIR	Direction signal which provides the rotating direction of the shaft encoder.
SAMPLE	Filtered sampling pulse used for latching the FPGA counter outputs.
ZM	Zero-marker signal.

In order to facilitate the data transfer between the FPGA and DSP, Cyclic Addressing circuitry is implemented in the FPGA, as shown in Fig. 4.4. A simple register addressing technique can be used because the data transfer between FPGA and DSP is performed in a predefined format for every sample interval, i.e. the

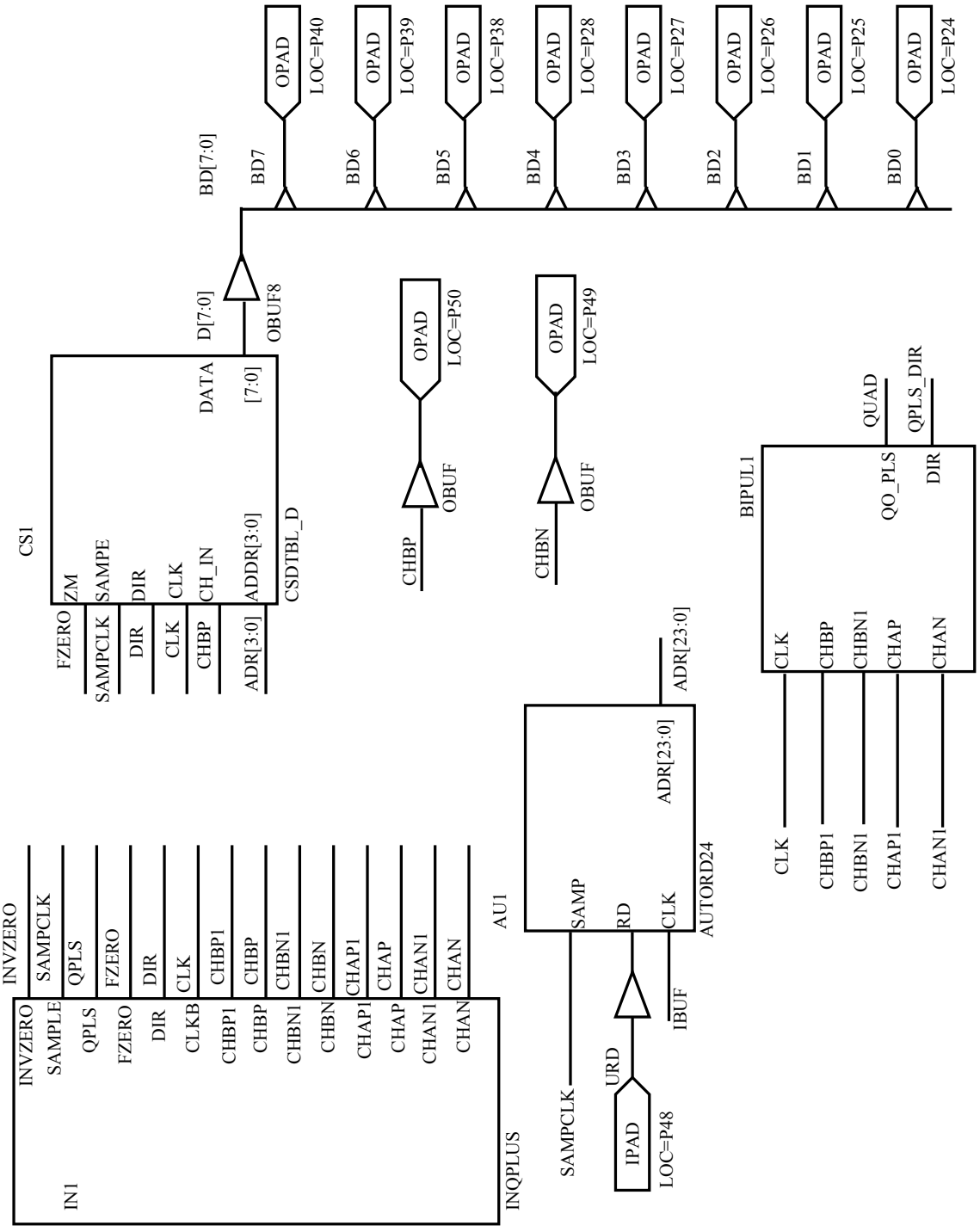


FIGURE 4.3: Block diagram showing the overall setup of the implementation.

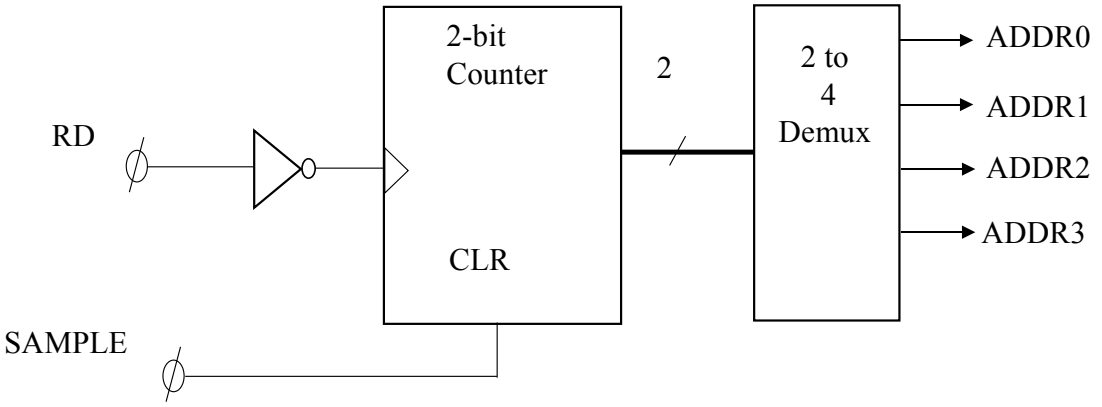


FIGURE 4.4: Block diagram of a typical FPGA/DSP-based servosystem controller (reproduced for convenience).

sequence of reading the FPGA register is the same after every sample instant. For convenience, a dSPACE DS1104 DSP card incorporating a 250 MHz 603 PowerPC floating-point processor is used to implement the learning and compensation algorithms, though a processor of significantly lesser power would often suffice. To investigate the implementation of tachometer-based closed-loop servosystems, the dSPACE software interface is used to implement a simple digital PI controller with a sample time, T_s, of 1 ms.

Three three-channel, optical incremental encoders were alternately mounted on two different permanent magnet synchronous motors of different sizes. One of the motor drives is connected to a flywheel load which ensures a smooth output velocity. As listed earlier in Section. 1.7.1, three distinct encoders are used. For convenience, Table 1.1 is repeated here as Table. 4.1.

To emphasize that the slit/interval errors exist in all kinds of encoders, encoder modules of different quality (and hence price) are selected for testing, with the Omron encoder being the most expensive of the three encoders, and the HP encoder

TABLE 4.1: Incremental encoder selected for experimental testing

Manufacturer	Resolution	No. of Channels
Omron	360	3
OVW	360	3
HP	200	3

being the cheapest.

4.2.1 Implementation of learning technique in open-loop

The implementation of the learning algorithm follows the multistage approach described in Chapter 2, where the encoder data (Pulse A, Pulse B and Zero-marker pulse) are captured, the encoder compensation table with transition position errors (easily derived from the interval width errors using $\delta_j = \delta_{j-1} + \delta_{(j-1),j}$, $j \in [1, L-1]$, with $\delta_0 = 0$), being computed off-line, for later use during normal operation. Several tens of thousands of data samples are acquired[1] (i.e. approximately over one minute) for the learning algorithm, though it is shown later that fewer samples will suffice. The learning time will be longer for higher-resolution encoders, as it then requires more samples to compute slit/interval errors. The algorithm can be implemented using MATLAB, or similar, in which case the stored CSDT data is passed to a computer and a fifth-order, zero-phase, low-pass Butterworth filter, with a cutoff-frequency chosen as 0.1 times the Nyquist frequency of the digital measurement system used to generate the reference velocity. When

[1]It was observed that the calculated shaft velocity contained error spikes due to the very occasional data mismatching caused by the asynchronous sampling of count/timing data within the FPGA. A simple procedure, similar to the double buffered method of Prokin [29, 30], is implemented to detect and remove such errors during both learning and normal operation.

using a zero-phase filter to remove the low-frequency oscillation from the CDST data, it is important to note that the x-axis corresponds to the encoder position, in place of the usual temporal axis, and the y-axis provides the line error, i.e. using position-based, rather than temporal units). A stand-alone DSP-based implementation, as shown in Fig. 4.1, was also tested. In comparison to the MATLAB implementation, where the encoder data is transfered and the compensation calculations performed off-line, the compensation calculations (in the DSP-based implementation using an iterative method) can be run as a background operation, while the servosystem continues to operate, without any interruptions, with pre-compensated velocity data. Given that the compensation algorithm (iterative method) can be implemented on-line, the memory requirement must also be much less than that for the MATLAB implementation. In this case, a third-order zero-phase filter, is used with a cut-off frequency similar to that used in the MATLAB implementation. While this is done for reasons of computational efficiency; it was found to result in minimal performance change relative to the fifth-order filter. In general, system performance was found to be insensitive to the precise cutoff frequency chosen, as highlighted by the fact that simple interpolation or averaging techniques were shown to give reasonable reference velocity waveforms, as will be described in more detail in Section 4.2.3.

4.2.2 Performance analysis of the proposed algorithms in an open-loop implementation

In [28] and Section. 3.4, results were presented of a comprehensive simulation of the encoder-based system in which the non-idealities of a typical encoder are modeled and then estimated using the iterative learning algorithm described in Chapter 2. The algorithm was shown to very accurately estimate the encoder errors. The results presented in this section are all based on experimental findings obtained while the motor drive is operated in open-loop.

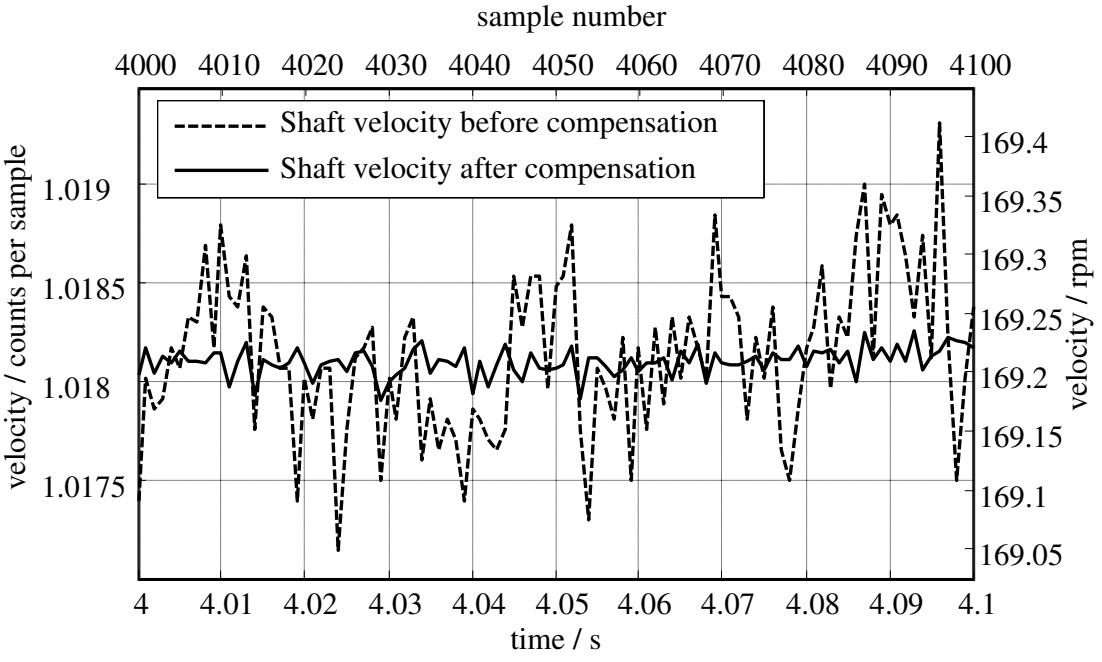


FIGURE 4.5: Estimated shaft velocity for Omron encoder (360 ppr) connected to high-inertia load, with MATLAB-based implementation of iterative learning algorithm.

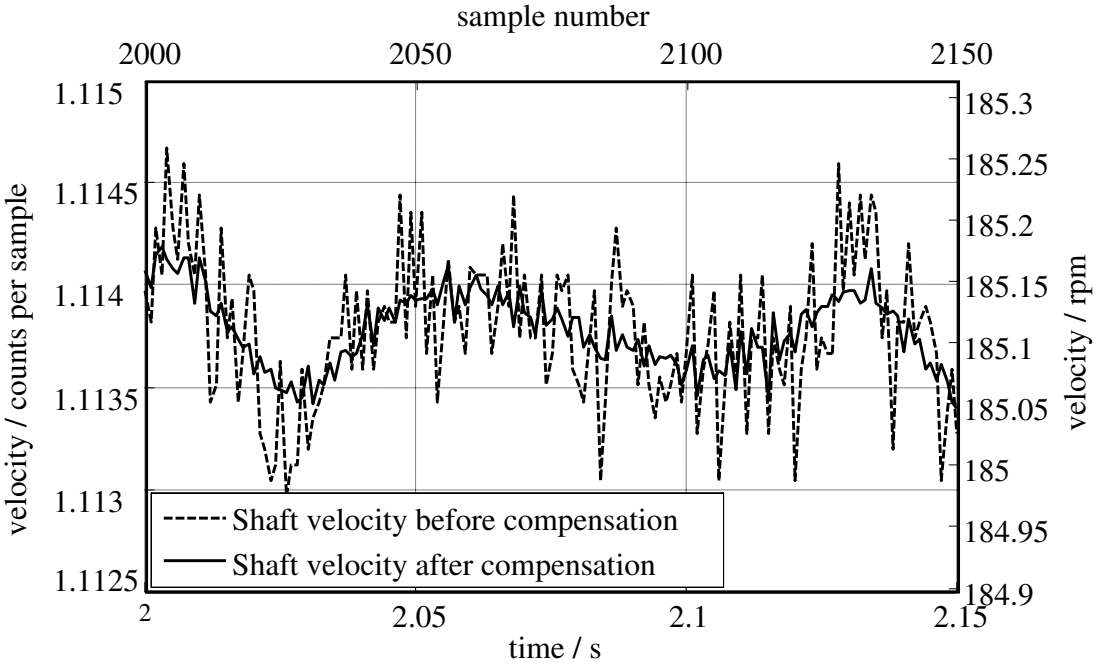


FIGURE 4.6: Estimated shaft velocity for OVW encoder (360 ppr), using DSP-based implementation of iterative learning algorithm.

4.2.2.1 Improvements in velocity estimates

The error reduction in CSDT velocity due to compensation is estimated by calculating the figure of merit, m, using (3.13). While the reference velocity, V_r, used in the calculation of m will not exactly match the actual average velocity over the measurement interval, it is essential that it provides a very good estimate; if anything, an error in V_r will tend to reduce the apparent benefit of compensation. From Fig. 4.5, where the comparison of encoder shaft velocities before and after the compensation are shown, the improvement in tachometer performance is clear for the Omron encoder that is tested. In order to validate the performance of the learning algorithms, similar tests are performed on the other types of chosen encoder (OVW and HP encoders). Performances obtained from the tests on these encoders, as shown in Figs 4.6 and 4.7, are similar to the performance with the

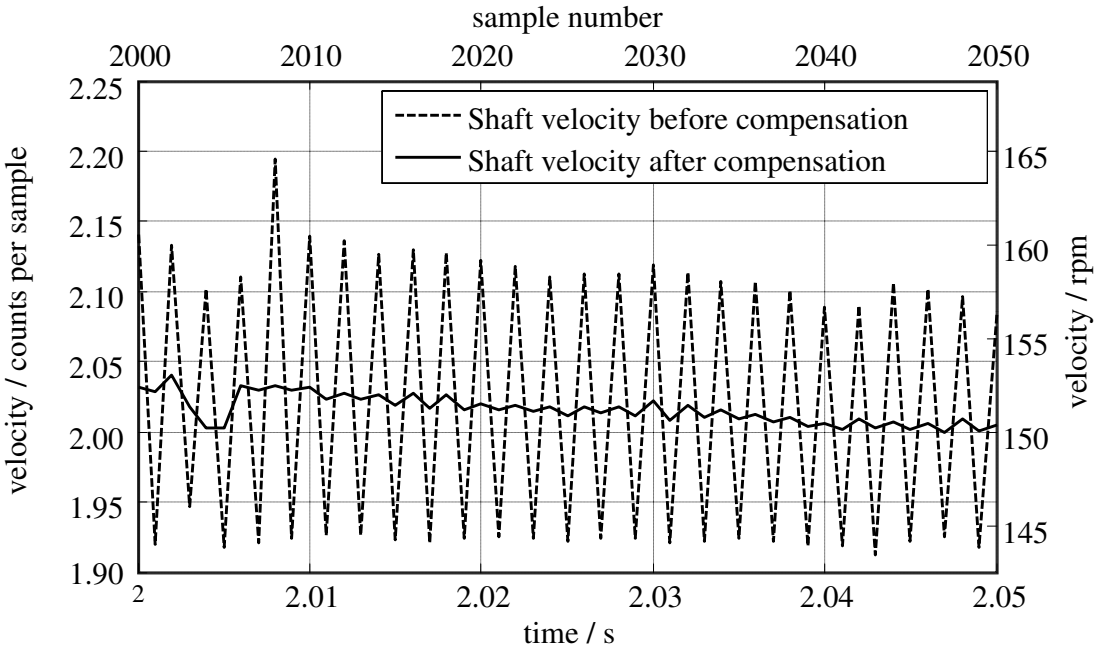


FIGURE 4.7: Velocities derived from flywheel-mounted quadrature decoded HP encoder signals; (MATLAB-based learning implementation). (Note: Two counts per sample time equates to 0.5 counts per sample interval without quadrature decoding).

Omron encoder. When the compensation routines are performed on quadrature-decoded signals, as in Fig. 4.7, it was found that the effect of compensation varies. For example, at speeds close to multiples of four transitions per sample interval, it is limited to that which would be expected if quadrature decoding had not been implemented. This is not a defect of the method, but reflects the fact that the edges being measured correspond to a particular transition (e.g. positive going transition of channel A) over many samples [71] and highlights the benefits of choosing judicious (preferably variable) shaft speeds during the learning phase.

In steady-state tests with approximately constant shaft velocity, the estimation errors were found to be reduced by over 80% in many tests. A subset of the generated data for the Omron and OVW encoders (connected to the high-inertia

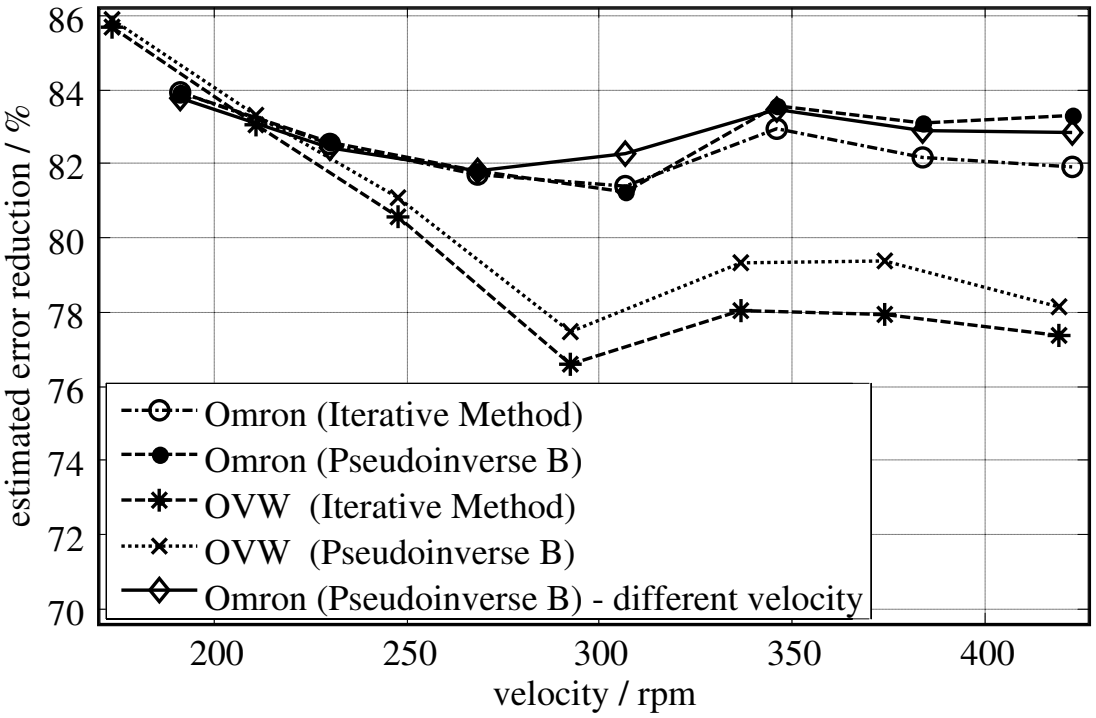


FIGURE 4.8: Estimated error reduction for high inertia system (using Omron and OVW encoders) at different velocities when using iterative and Pseudoinverse B algorithms, with look-up tables calculated at the same speeds at which the compensation is performed, with the exception of the trace marked with '◇', for which the look-up table is calculated using a ramp-down traverse from 460 rpm to 310 rpm.

load, which also limits the maximum speed) is presented in Fig. 4.8. The proposed algorithm is also tested using the low cost, modular Hewlett Packard encoder connected to a low-inertia shaft (so that the velocity will not be so smooth). The compensation algorithm is found to work almost equally well; see Fig. 4.9. Some of the traces in these figures refer to tests in which the data used was acquired at a similar velocity to that being measured. However, two traces in Figs. 4.8 and 4.9 refer to data obtained at different velocities to those used for the tests. These are intended to investigate the effects of the inevitable changes in the encoder characteristics as shaft velocity varies, due to the changing lags and transfer functions of analog circuitry within the encoder [72]. While the variation may be small,

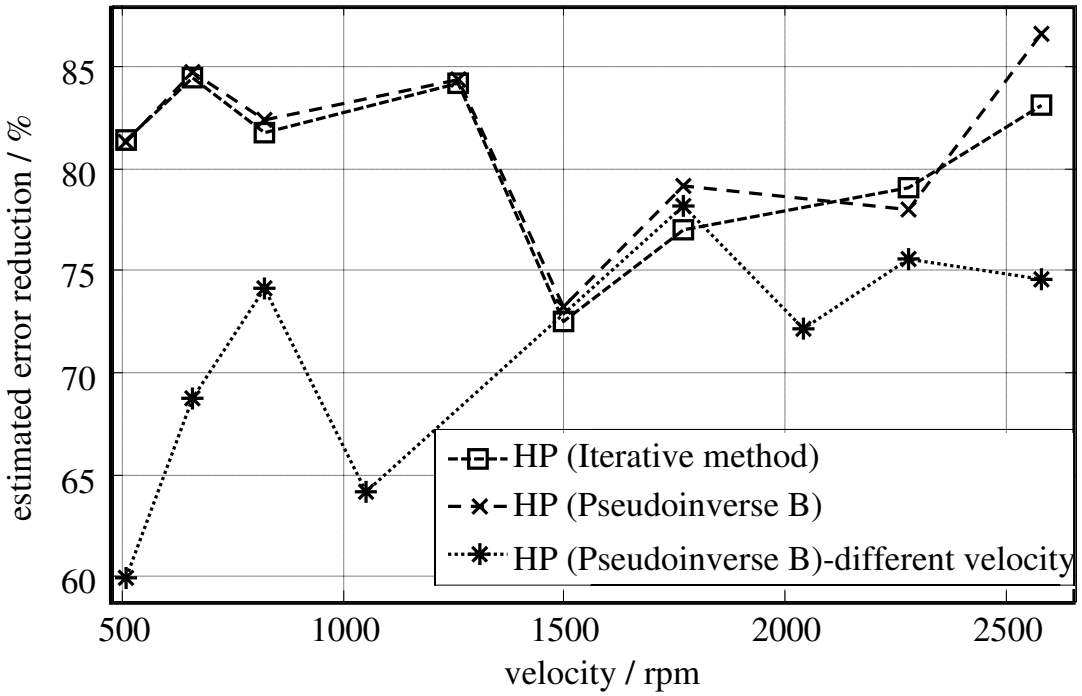


FIGURE 4.9: Estimated error reduction for low inertia system (using HP encoder) at different velocities, with look-up tables calculated at same velocities at which the compensation is performed, with the exception of the trace marked '*', where the look-up table is calculated during a ramp-up traverse from 2040 rpm to 2310 rpm.

it could conceivably have a significant effect on the benefits of compensation because any change directly affects the encoder line positions, unlike other, perhaps large, noise components in the system. This points to the advisability of storing a number of compensation tables (which should be generated, preferably, when the shaft velocity is varying). The relevant trace in Fig. 4.9 represents a worst-case scenario, as high-speed compensation data is used for low-velocity compensation, but significant benefit still accrues from compensation. It is more logical to obtain data at lower velocity, where measurement accuracy is usually most crucial.

In order to evaluate the performance of the method proposed in this thesis against a similar method, that proposed by Merry et. al. [4, 5] (presented in Chapter 1),

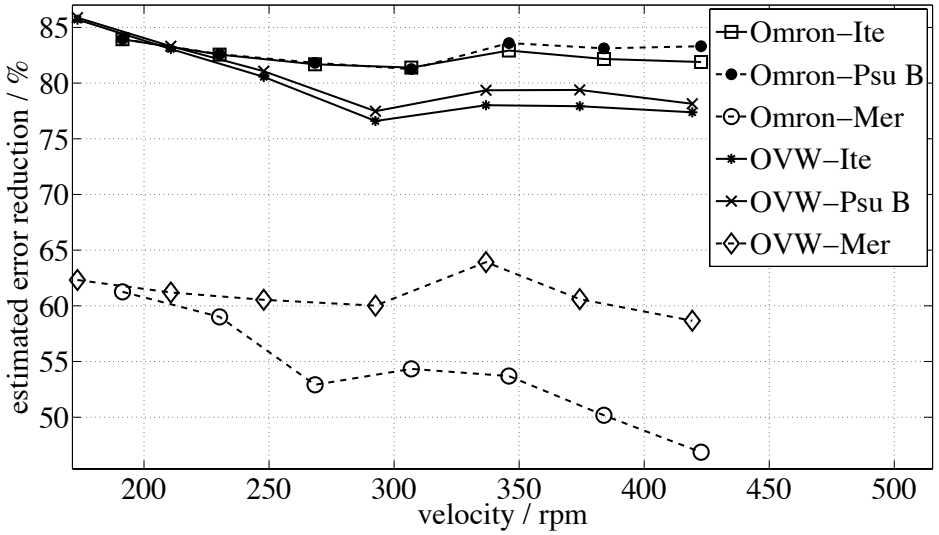


FIGURE 4.10: Estimated error reduction for high inertia system (using Omron and OVW encoders) at different velocities when using iterative (Ite), Pseudoinverse B (Psu B) and Merry's [4, 5] algorithms (Mer), with look-up tables calculated at the same speeds at which the compensation is performed

is selected. The comparison between the proposed Iterative and Pseudo-inverse B methods and a method based on Merry's is shown in Fig. 4.10. It is clear that the new methods significantly out-perform that of Merry.

4.2.2.2 Performance and output of the learning algorithm

It is clear from Figs. 4.8 and 4.9 that the iterative algorithm works almost as well as the mathematically more complex pseudoinverse solution to line errors. Comparisons between the three solution techniques described in Chapter 2, as a function of sample length, N_s, are shown in Figs. 4.11, 4.12 and 4.13 for the Omron, OVW and HP encoders respectively. The figures indicate that all the methods show similar patterns of convergence, though the sample size must be larger for the iterative method, as might be expected. It is seen that $N_s = 10,000$ is sufficient for a 360 line encoder. The two pseudoinverse-based methods are

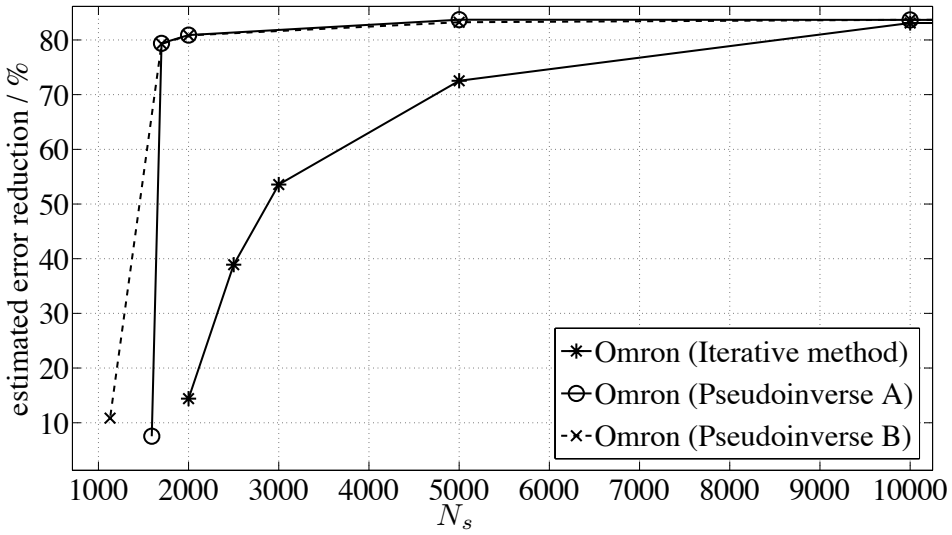


FIGURE 4.11: Calculation of estimated error reduction (using Omron encoder with 360 ppr) at 170 rpm, for various sample lengths, N_s.

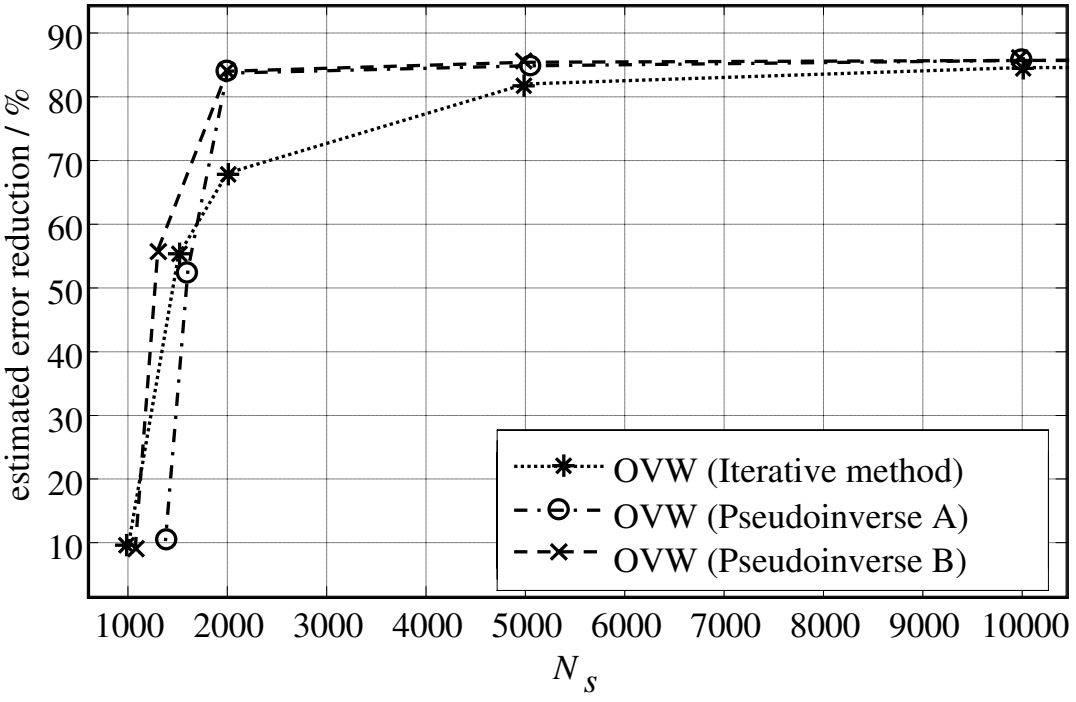


FIGURE 4.12: Calculation of estimated error reduction (using OVW encoder with 360 ppr) at 170 rpm, for various sample lengths, N_s.

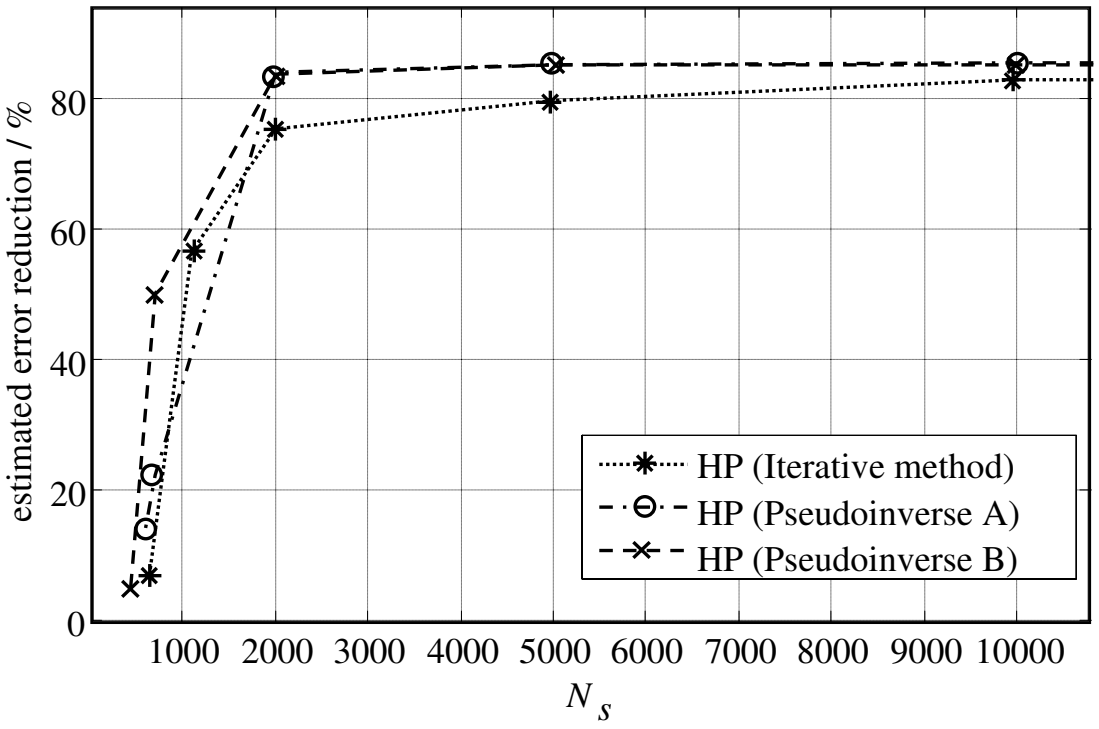


FIGURE 4.13: Calculation of estimated error reduction at 1700 rpm for HP encoder (200 ppr), without flywheel, as N_s varies.

equivalent, except for very small data sets, in which case the Pseudoinverse A method is preferred.

In a further test, the learning algorithm was applied to both edges of the two channels (CH-A and CH-B) in clockwise and anti-clockwise directions and the slit errors calculated. As shown in Fig. 4.14, the slit errors estimated for the positive edge of channel B in the clockwise direction are similar to those obtained when the negative edge of channel B is considered in the anti-clockwise direction, as it should be. Similar results are observed with channel A. The convergence traces for some arbitrarily selected interval errors, $\delta_{j(j+1)}$, calculated using the iterative learning process, are shown in Fig. 4.15.

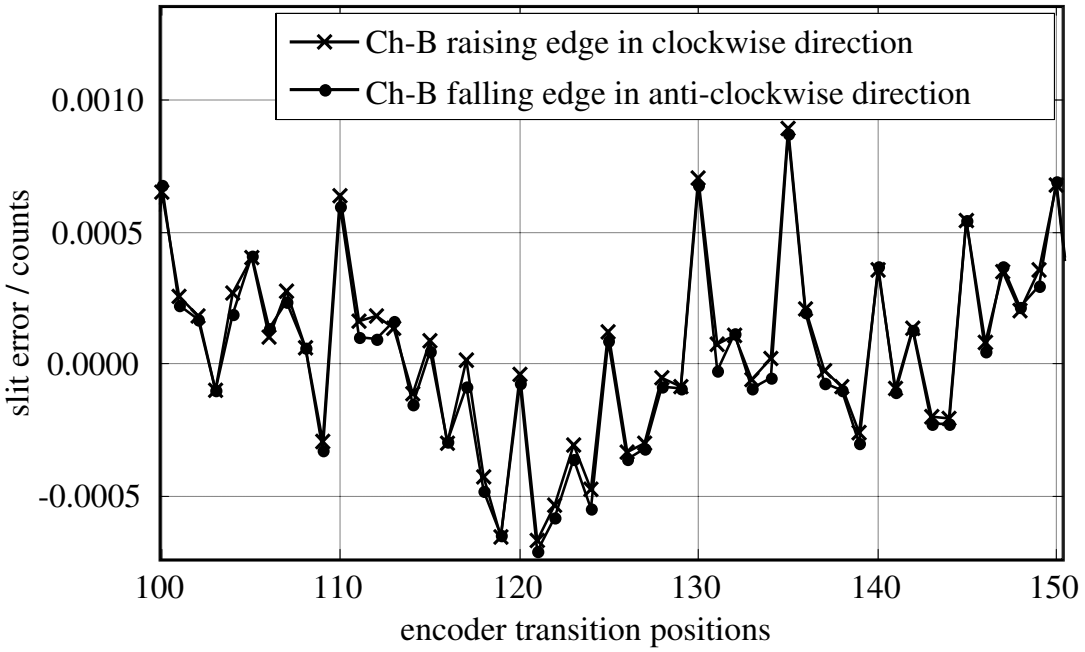


FIGURE 4.14: Slit errors of OVW encoder's transition positions 100 to 150 for positive-going edge of channel B in clockwise, and negative-going edge in anti-clockwise, directions, at 170 rpm.

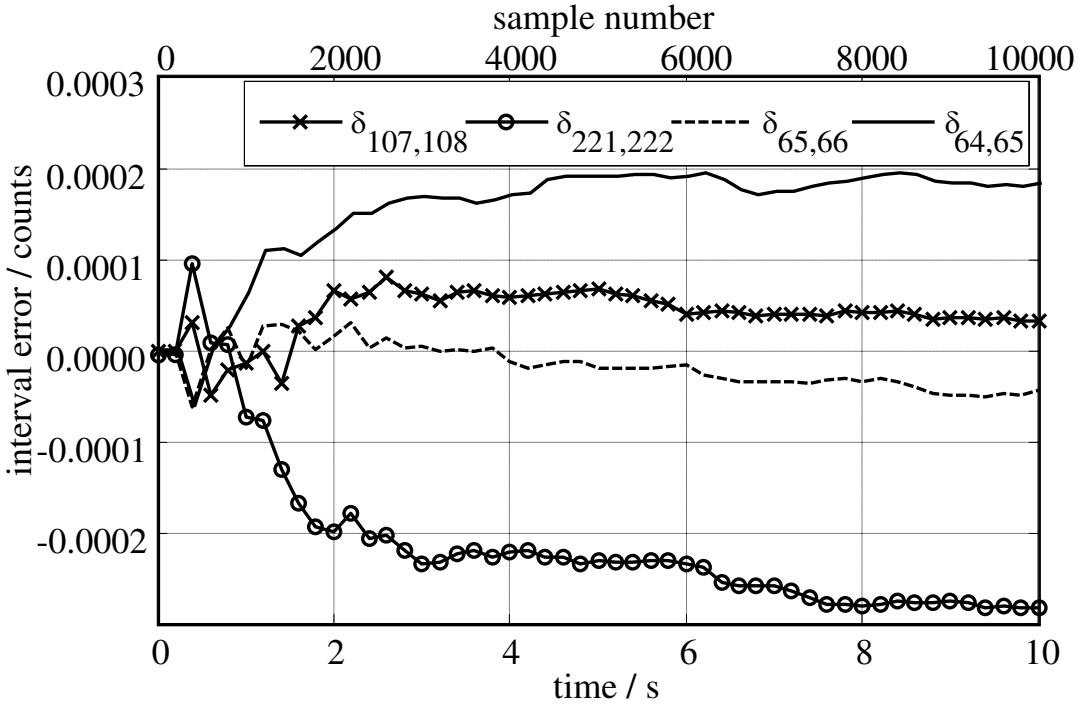


FIGURE 4.15: Convergence of interval error estimates using the iterative learning algorithm, to steady-state values, for a few randomly selected interval errors at 170 rpm, using an Omron encoder, with 360 ppr.

4.2.3 Investigation of alternative means of calculating the reference velocity

The computation of reference-velocity values described in Section 3.4.1, and the consequent results presented above, are based on the generation of a reference velocity, V_r, waveform using a zero-phase filter. Error in V_r, system noise, and unmodeled encoder non-idealities will all influence the **e** error vector described in Chapter 3. However, given sufficient samples, the encoder errors can be estimated well using (2.12), or any of the other methods described above, when the $e(i)$ values are uncorrelated with the relevant encoder errors, i.e. are uncorrelated with $\delta_{P(i)}$ and $\delta_{P(i-1)}$, or $\delta_{k,(k+1)}$, $k \in [P(i-1), P(i)-2]$.

The zero-phase filter is found to provide an excellent reference velocity. In order to facilitate real-time encoder compensation, some simpler methods of generating $V_r(i)$ are considered. Other possibilities are to use a variant of the polynomial-fit method of Merry *et al.* [4, 5], or a simple average of a number of CSDT velocity values before and after sample i, as explained in Section. 3.4.1. It is clear from Fig. 4.16 that the results from the polynomial-fit and the mean velocity calculation methods are inferior to those due to application of the zero-phase filter. However they still allow significant performance improvement. The fact that a lag of only approximately half the number of samples used in the velocity reference estimate exists between taking data, and being able to then start processing that data for encoder compensation, is a significant advantage which allows the iterative algorithm to be incorporated more easily in a real-time controller.

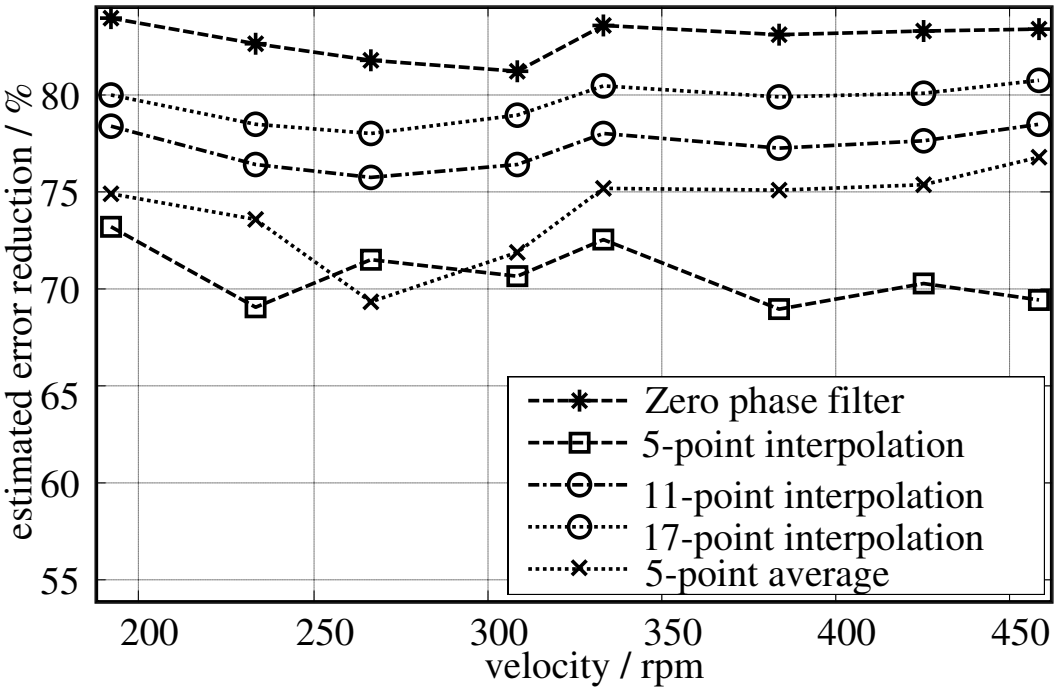


FIGURE 4.16: Estimated error reduction, using Pseudoinverse B method with Omron encoder (360 ppr), for variously calculated reference velocities.

It is important to remember that for the CSDT data, angular distance is the x-axis variable. Filtering of this data usually utilizes a zero-phase filter, the nyquist frequency of which is chosen. Therefore, a desired low-pass filter bandwidth is dependent on the number of adjacent encoder transition position over which low-pass filtering (averaging) will be employed.

In order to analyze the effect of the chosen bandwidth of the zero-phase filter on high-frequency error calculation, several tests using different filter bandwidths were conducted. Note that the bandwidths specified are normalized relative to the Nyquist frequency. This, in turn, corresponds to a frequency of $L/2$ per revolution. The obtained root-mean-square of the high-frequency error in the compensated shaft velocity was obtained in each case. The results indicate that the resultant

learning errors are insensitive to the chosen bandwidth of the zero-phase filter used (as shown in Fig. 4.17).

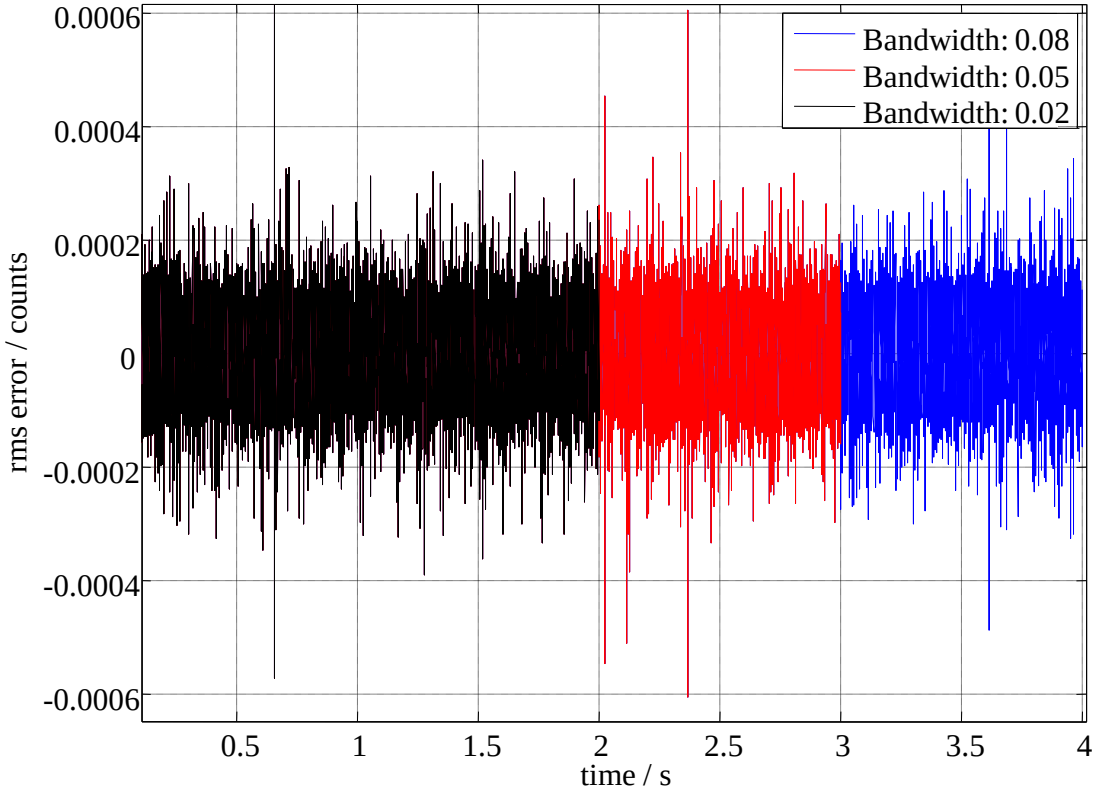


FIGURE 4.17: Root-mean square of high-frequency error in the compensated shaft velocity at three different band-widths obtained by implementing the Pseudoinverse-A learning algorithm on the Omron encoder.

4.2.4 Experimental evaluation of the noise and errors associated with an encoder

To match the theoretically postulated encoder non-linearities with those of a real encoder, a comparision of the probability density functions of interval and slit errors are performed. As shown in Fig. 4.18, the interval and slit errors of an Omron encoder, with a resolution $N = 360$, are calculated (using the iterative

learning algorithm). The probability density function of the interval error is almost triangularly distributed, while the slit error, which is often assumed to be uniformly distributed, is not exactly so. This can be caused by the nature of the encoder wheel and electronics, or by any imperfections in determining the interval error. However, the results obtained from experimental data are consistent with the behaviour of a simulated encoder, as shown in Fig. 3.7, based on the theoretical model of a typical encoder, as described in Chapter. 3.

In order to demonstrate the influence of the (encoder) position-dependent random noise, $\delta^n_{P(i)}$, and position independent random noise, $\delta^n(i)$, on the performance of the learning algorithm, a comparative study is performed between the mathematical model and experimental implementation. The probability density function and figure of merit of the learning algorithm and the standard deviation of the interval error were considered as important criteria. When comparing simulated and experimental devices, it will be assumed that the two show a great similarity when the figure of merit and the standard deviation are both similar. As shown in Fig. 4.19, the probability density function of different, but reasonable, mathematical model scenarios (as documented in Table. 4.2), are found to approximate well to the experimental data, based on the chosen criteria. The simulated data is are plotted against the p.d.f. of the interval error obtained from the Omron encoder (360 ppr). The experimental and model data are shown to be almost identical. For analysing the various possibilities, the position-dependent and position-independent noises (having the same standard deviation as the Omron encoder) in some chosen mathematical models (with different parameters) are

compared to experimentally obtained results. From Fig. 4.19 and Table. 4.2 it can be concluded that the results are consistent with the assumptions made in the mathematical modelling. Hence it can be concluded that assumptions made in modelling the encoder errors are reasonable and insightful.

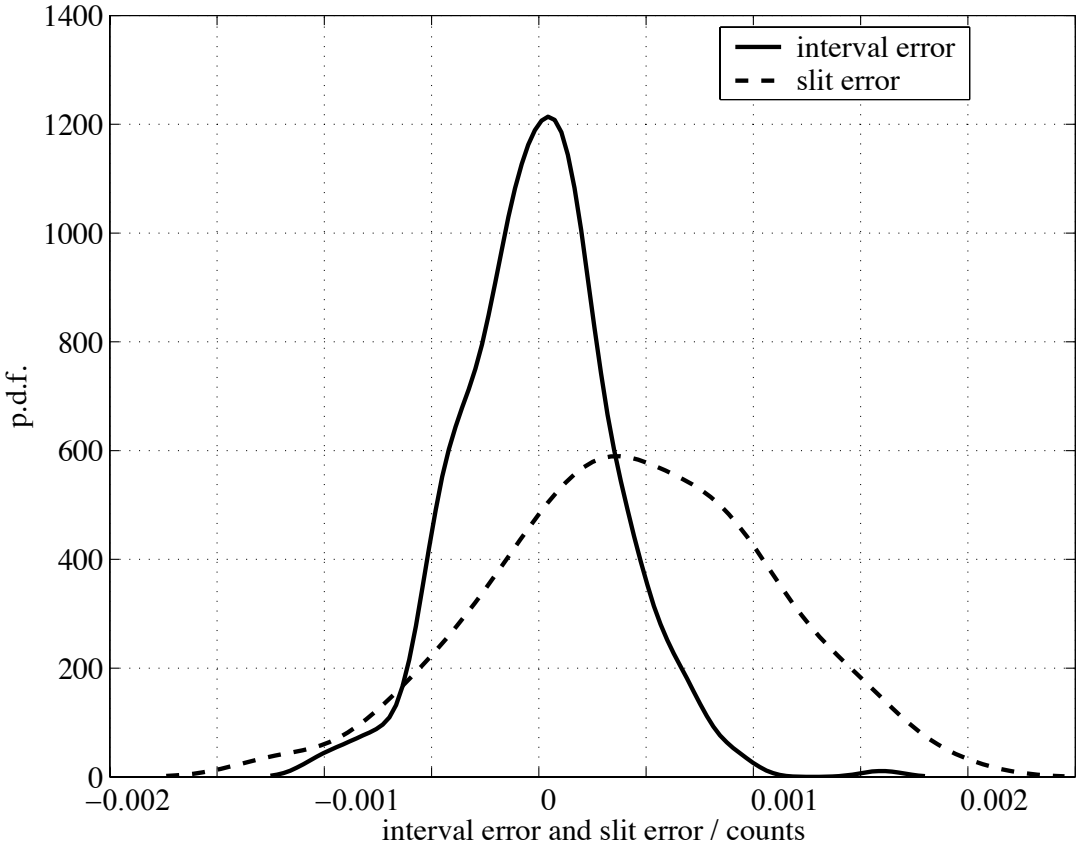


FIGURE 4.18: Probability density function of interval and slit errors of the Omron 360 ppr incremental encoder, calculated using the iterative learning algorithm.

However, a detailed breakdown of the error source is (a) not possible without a very large number of tests and calculations, and (b) will vary with different encoders. Therefore, such over-analysis is not particularly helpful, given that the application of the learning algorithm will work well without generation of such a detailed model.

TABLE 4.2: Comparative study of observed experimental characteristics and judiciously selected mathematical models performance

Type of test	Average position dependent noise / count ($\times 10^{-4}$)	Average position independent noise / count ($\times 10^{-5}$)	Standrad deviation ($\times 10^{-4}$)	Figure of merit
Experimental	–	–	3.7452	84.00
Simulation	2.5647	3.1774	3.8749	84.06
Simulation	2.2560	2.8853	3.5256	83.63
Simulation	1.9185	1.5880	3.5570	84.12
Simulation	1.9792	1.6410	3.0780	84.20

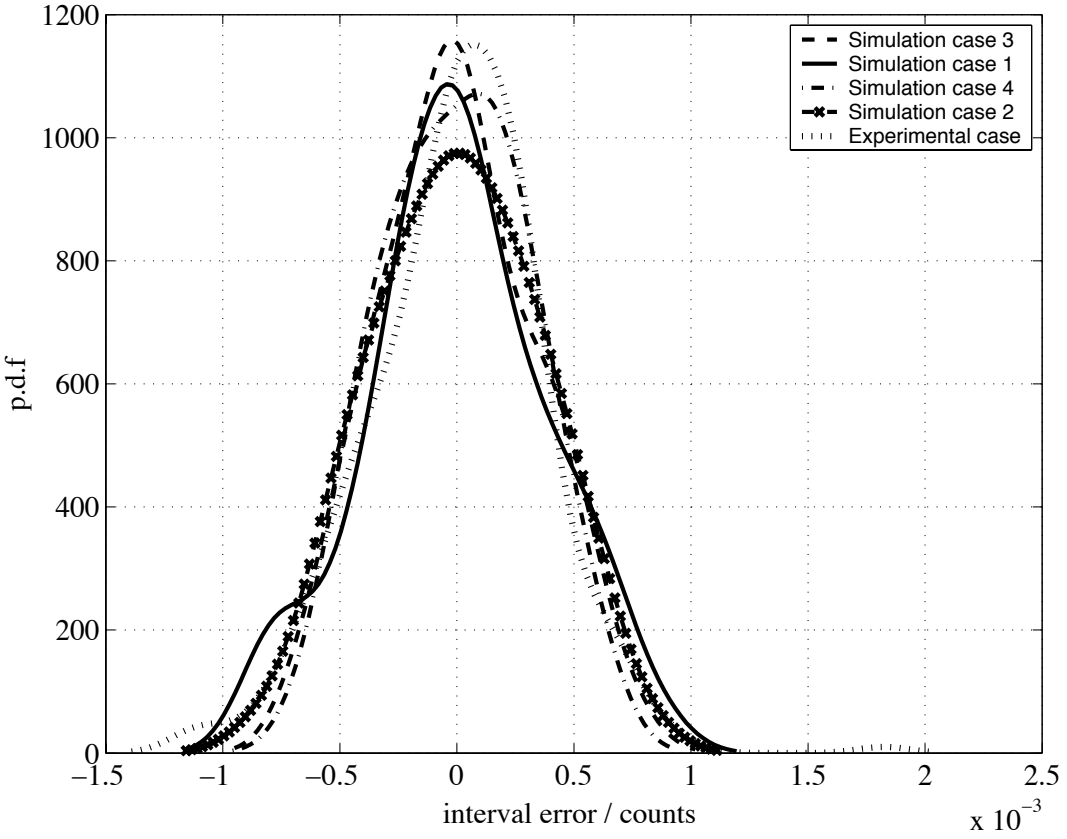


FIGURE 4.19: Probability density function of interval error for Omron incremental encoder, and comparison with some mathematical models (with resolution of 360 ppr, calculated using iterative learning algorithm).

4.3 Experimental Results for a Closed-Loop Servosystem

The proposed learning algorithm was incorporated into a closed-loop servosystem, thereby allowing investigation of effects such as delay in the feedback provided by the CSDT, and of sensor discretization and sampling, on the stability of the servosystem. The effects of delay leading to self-sustained oscillation are well explained by Kuo and Kubis in [73, 74]. A linear, torque-mode, high-bandwidth amplifier driving a two-pole, permanent-magnet synchronous motor (PMSM) without flywheel load and the 200 ppr, modular Hewlett Packard encoder are used in this work, along with the previously described FPGA/DSP combination for CSDT and controller implementation.

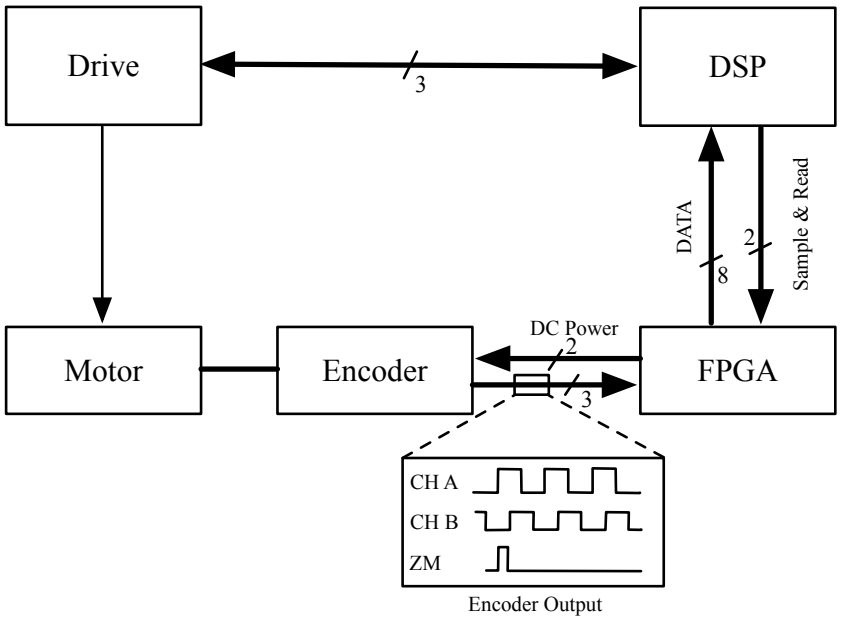


FIGURE 4.20: Block diagram of a typical FPGA/DSP-based servosystem controller (reproduced for convenience from Section 1.7).

4.3.1 Design of the closed-loop controller

TABLE 4.3: Servosystem Parameters used for Experimental Implementation

System Parameters	Value
DAC gain, K_{dac}	10 V/(unit controller O/P)
Amplifier gain, K_a	2 A/V
Torque constant, K_t	0.061 Nm A^{-1}
Encoder gain, $K_{enc} = L/(2\pi)$	500 / (2π) cycles rad^{-1}
Motor Inertia, J	59×10^{-6} kg m^2
Sample Time, T_s	1 ms

The controller is designed using the method of Roche *et al.* [75]. To obtain a simplified plant model, it is assumed that the mechanical linkage between motor and driven load has infinite stiffness, viscous friction on the mechanical load is negligible, and that the power amplifier and motor act as an ideal torque actuator. The parameters of the servosystem are provided in Table 4.3.[2] Using the root locus technique, the proportional and integral gains of the PI controller, K_{vp} and K_{vi}, can be calculated, for a damping ratio of 0.707, as

$$K_{vp} = \frac{0.06868}{K_{pl}{T_s}^2} \tag{4.1}$$

$$K_{vi} = \frac{0.3431}{K_{pl}{T_s}^2} \tag{4.2}$$

where the plant has gain

$$K_{pl} = \frac{K_{dac}K_aK_tK_{enc}}{J}, \tag{4.3}$$

from which the approximate values of proportional gain K_{vp} and integral gain K_{vi} are calculated as 0.208 and 0.0417, respectively.

[2]The output of the PI controller calculation is a fractional number that is converted to an amplifier voltage input via a DAC, the gain of which is represented as K_{dac}.

The integration of CSDT into the control structure must be done with care. Velocity information from both the CSDT and PCT (position difference) are used in tandem at low speeds, to ensure correct output from the integral part of the controller. The stability of the CSDT-enhanced controller is equivalent to that of the usual PCT-based controller, but with much-improved accuracy at speeds greater than one count per sample interval.

4.3.2 Comparative results to verify utility of the learning algorithm in a closed-loop system

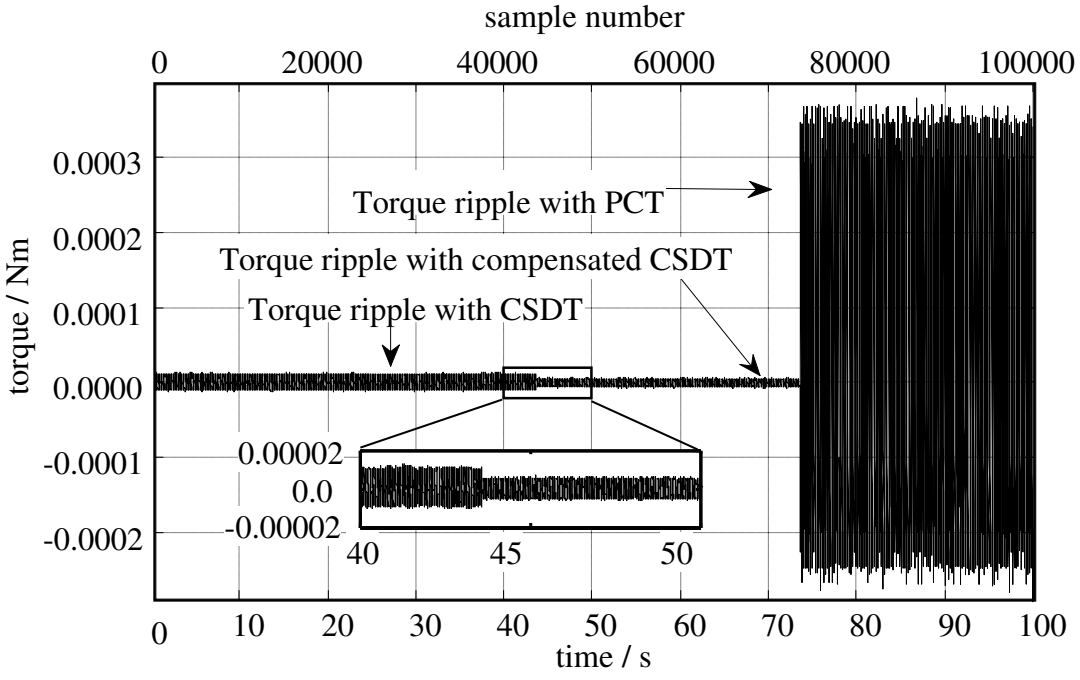


FIGURE 4.21: Torque ripple, showing effect of switching velocity feedback source from uncompensated CSDT, to compensated CSDT and, finally, to PCT, when using 200 ppr HP encoder rotating at 2000 rpm, (the d.c. motor torque is 0.0294 Nm approximately).

In [75], rms current ripple was used to evaluate the noise introduced into the system due to the control algorithm, physical characteristics of the servosystem

and quantization error. In this thesis, the root-mean-squared (rms) value of the high-frequency components of input torque ripple is instead used to compare the closed-loop performance of the different digital tachometers at a randomly selected speed of 2000 rpm. The results, normalized with respect to that due to use of a pulse-count tachometer (PCT) without quadrature decoding, are shown in Table 4.4. The PCT induces the largest ripple content in the input torque, predominantly due to quantization error and, to a lesser extent, encoder transition location errors. The torque ripple due to compensated CSDT-based velocity feedback is approximately five times less than the equivalent value without compensation (which is typical). A comparison of torque ripple pertaining to use of the PCT feedback and that corresponding to use of either a compensated or uncompensated CSDT is shown in Fig. 4.21.

TABLE 4.4: Normalized RMS Value of High-Frequency Torque Ripple in Servosystem

Type of tachometer	Normalized h.f. trq. ripple (rms)
PCT	1
CSDT	0.0119
CSDT (compensated)	0.0022

Though an error reduction of over 95% was observed in the various simulations consistently for different velocity profiles, the high-frequency error reduction observed in the experimental implementation is around 80%. As the influence of quantization error is mostly eliminated by using the CSDT velocity measurement techinique (given that the sample time T_s is 1 ms and the high speed clock frequency is 20 MHz, the typical error due to quantization while implementing the CSDT method will be in the order of, $\frac{1}{f_s \cdot T_s} = \frac{1}{20000}$), this deviation between the

simulation and experimental results can be assumed to be mostly due the random noise generated in the experimental system and the imperfect realization of the calculated reference velocity, when compared to actual reference velocity.

Chapter 5

Conclusions

5.1 Conclusions

Three learning algorithm are proposed as a means of estimating the slit errors (i.e. transition location errors) or interval error in the code wheel of a square-wave incremental shaft encoder. A trivial modification of the methods would enable it to be used equally for absolute encoders. While some previous work in this field has shown the ability to reduce encoder error by 77% to 80%, these implementations have all had significant constraints, requiring the use of high-specification reference sensors or special fixtures and/or hardware. This work proposed three different learning methodologies to estimate the interval and slit errors. Of them, the Pseudoinverse-A and Iterative methods are dedicated to computing slit errors, while the Pseudoinverse-B algorithm estimates the interval errors of the encoder. Under varied conditions, improvements up to 86% (but typically approximately 80%) in shaft velocity measurement, when compared to velocity measurement

using the CDST method, are reported in this work. It is notable that the improvements are consistent for encoders with different resolutions from different manufactures and for different traverses (some noisier than others). The chosen CSDT velocity measuring method is shown to provide sufficient data to enable implementation of the learning algorithms. Results highlight the possibility of using a stand-alone implementation in which an iterative learning algorithm can be used (possibly retrofitted), while a pseudoinverse-based off-line solution was shown to provide the best performance with fewer data samples. The application of a zero-phase filter to the CSDT waveforms was seen to provide an excellent 'reference' velocity. However, the demonstrated viability of using a simpler method, such as interpolation and average methods, to generate the waveform, was shown to ease real-time implementation with only slightly reduced performance.

It is clear that the compensation is a once-off, or very occasional, operation that can function without regard to the specific control algorithm, or the drive or load parameters, and its application does not disallow control structures such as an observer or a Kalman filter. Given the lack of a reference sensor and the reliance on the internally generated reference velocity, it makes sense to implement the learning phase when the drive is somewhat detuned, or at least when it is not exciting resonances in the load. This is easily arranged in most practical applications. The stipulation that effective operation of the learning algorithm implies that those principal velocity error components measured by the CSDT which are correlated with the shaft position are due to encoder transition position errors should be borne in mind, particularly when drives with gearboxes are involved.

The precision of velocity measurement with the pulse count and CSDT methods, with and without compensation, are repeated in Table. 5.1 for convenience.

TABLE 5.1: Normalized RMS Value of High-Frequency Torque Ripple in Servosystem

Type of tachometer	Normalized h.f. trq. ripple (rms)
PCT	1
CSDT	0.0119
CSDT (compensated)	0.0022

The proposed algorithms can be applied in many application such as accurate trajectory following in very high bandwidth robotics applications, numerically-controlled (NC) machines, etc.

Due to the limited time and resources available, certain limitations were set for the project. All the algorithms proposed are tested only with optical incremental square-wave encoders. In terms of potential future work, a number of extensions are possible to the work presented in this thesis. While the learning techniques have been designed in such a way that they can be utilized with little human intervention, further work should be undertaken on how and when the servosystem should self-calibrate, i.e. based on the trajectory that the servo is undertaking, or indeed, a decision may be made that calibration is only required during the installation phase of an automation system. In closed-loop systems, it would be possible to augment the digital tachometer with a non-linear observer [76, 77]. With relatively minor alterations, it would also be possible to implement the learning-algorithm on absolute encoders. Using the algorithm on sinusoidal encoders would effectively take into account the fact that different actual widths

are associated with different electrical cycles of the encoder output. This would require an algorithm that utilized both the line width learning techniques presented in this thesis, and the learning method of Kavanagh [44] for the average shape of nominally sinusoidal waveforms in sinusoidal encoders. There does not appear to be any major impediment to implementing such a system. The work described in this thesis has been reported in a conference publication [28] and in the IEEE Transactions on Instrumentation and Measurement [78].

Appendix A

Architecture of DS1104 dSPACE R & D Controller Board

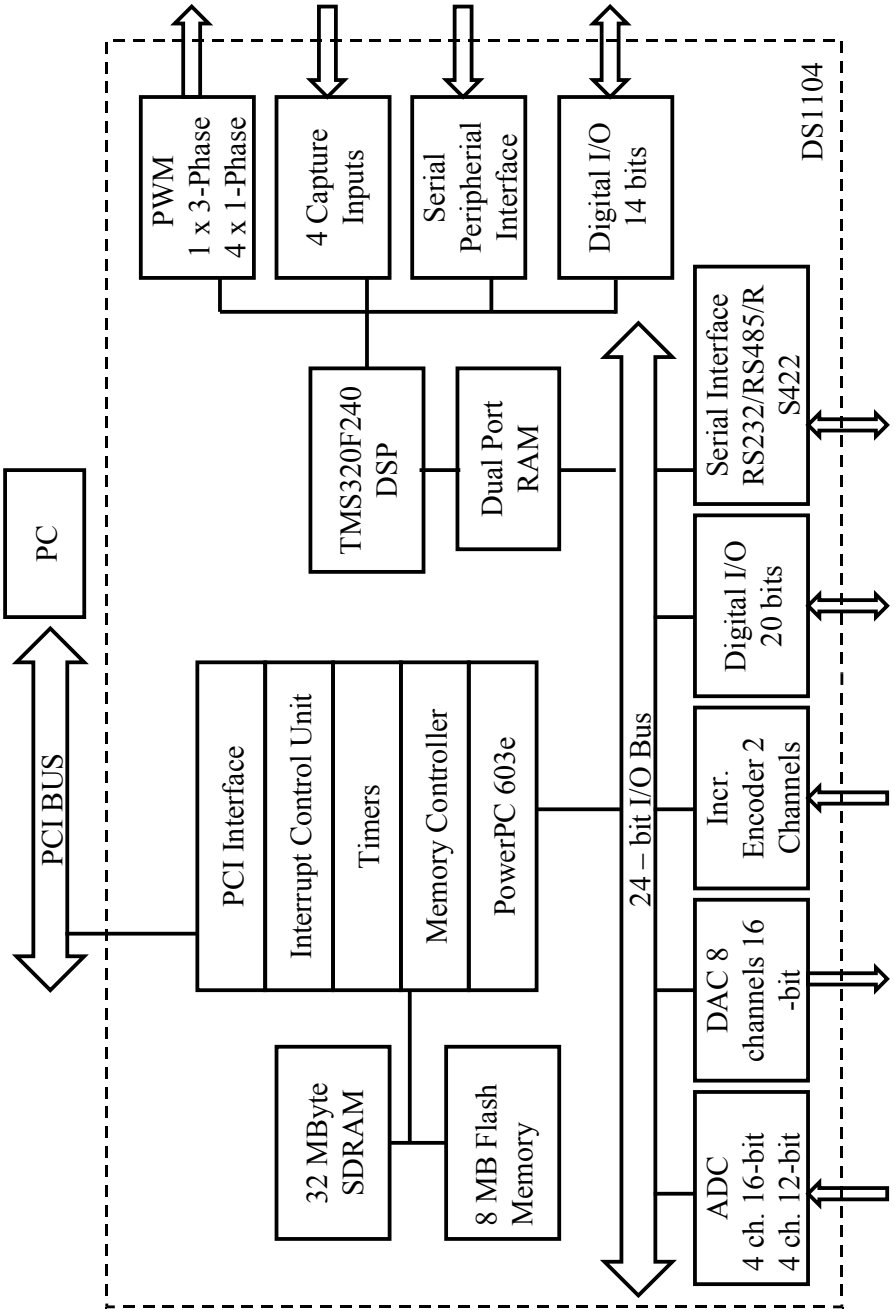


FIGURE A.1: DS1104 R & D Controller Board Block Diagram.

Bibliography

[1] C. Yien. Incremental encoder errors: Causes and ways to reduce them. In *Proc. PCIM'92*, pages 110–121, Nuremberg, Germany, Apr. 1992.

[2] R. Nutt. Digital time intervals meter. *Rev. Sci. Instruments*, 39:1342–1345, 1968.

[3] G.Strang. *Linear algebra and its applications*. Harcourt Brace Jovanovich, United States of America, third edition, 1988.

[4] R. Merry, R. van de Molengraft, and M. Steinbuch. Error modeling and improved position estimation for optical incremental encoders by means of time stamping. In *American Control Conference*, pages 3570–3575, New York, USA, 2007.

[5] R. J. E. Merry and M. J. G. van de Molengraft and M. Steinbuch. Velocity and acceleration estimation for optical incremental encoders. In *The Intl. Federation of Automatic Control*, pages 7570–7575, Seoul, Korea, Jul. 2008.

[6] Saeed B. Niku. *Introduction to Robotics - Analysis, Control, Applications*. John Wiley and Sons, Inc., United States of America, 2011.

[7] M.C. Srivastav M. Srivastava and S. Bhatnagar. *Control Systems*. Tata McGraw-Hill publications., NewDelhi, India, 2009.

[8] E. A. Parr. *Industrial Control Handbook*. Butterworth-Heinemann Publications, Oxford, U. K., 1989.

[9] J. Lenarcic A. Stanovnik T. Bajd, M. Mihelj and M. Munih. *Intelligent systems, control, and automation: Science and engineering*. Springer Publications, London, U. K., 2010.

[10] T. R. Padmanabhan. *Industrial instrumentation: Principles and design*. Springer-Verlag Publications, London, U. K., 2000.

[11] W. Boyes. *Instrumentation reference book*. Butterworth-Heinemann Publications, London, U. K., 2003.

[12] G. H. Ellis. *Control system design guide: A practical guide*. Elsevier Academic Press, London, U. K., 2004.

[13] M. Leonard. Push-pull optical detector integrated circuit. *IEEE Jnl. Solid State Circuits.*, SC-15:1087–1089, Dec. 1980.

[14] H. Epstein, M. Leonard, and R. Nicol. Economical high-performance optical encoders. *Hewlett-Packard Journal*, 39(5):99–106, Oct. 1988.

[15] R.M. Gray. Quantization noise spectra. *IEEE Transactions on Information Theory*, 36(6):1220–1244, Nov. 1990.

[16] S. R. Norsworthy, R. Schreier, and G. C. Temes. *Delta-Sigma Data Converters-Theory, Design and Simulation*. IEEE press, 1992.

[17] Gene F. Franklin, J. David Powell, and Michael L. Workman. *Digital Control of Dynamic Systems*. Addison-Wesley Publications & Co., third edition, 1997.

[18] K. W. Cattermole. *Principles of Pulse Code Modulation*. Iliffe Books Ltd., 1969.

[19] R. C. Kavanagh and J. M. D. Murphy. The effects of quantization noise and sensor non-ideality on digital-differentiator-based velocity measurement. *IEEE Trans. Instrum. Meas.*, 47:1457–1463, Dec. 1998.

[20] G. A. Woolvet. *Transducers in Digital Systems*. Institution of Electrical Engineers, London, U.K, 1977.

[21] T. Ohmae, T. Matsuda, K. Kamiyama, and M. Tachikawa. A microprocessor-controlled high-accuracy wide-range speed regulator for motor drives. *IEEE Trans. Ind. Electron.*, 29(3):207–211, Aug. 1982.

[22] R. C. Kavanagh. Improved digital tachometer with reduced sensitivity to sensor non-ideality. *IEEE Trans. Ind. Electron.*, 47(4):890–897, Aug. 2000.

[23] J. Mayrhofer and J. Strachwitz. A control circuit for electrical drives based on transputers and FPGAs. In W.Moore and W.Luk, editors, *More FPGAs*, pages 420–427. Oxford, Abbington EE and CS Books, 1993.

[24] J. Kalisz, M. Pawlowski, and R. Pelka. Error analysis and design of the Nutt time-inverval digitiser with picosecond resolution. *J. Phys. E, Sci. Instrum.*, 20(11):1330–1341, 1987.

[25] J. Kalisz, R. Szplet, R. Pelka, and A. Poniecki. Single-chip interpolating time counter with 200-ps resolution and 43-s range. *IEEE Trans. Instrum. Meas.*, 46(1):51–55, Feb. 1997.

[26] J. Kalisz, R. Szplet, J. Pasierbinski, and A. Poniecki. Field-programmable-gate-array-based time-to-digital converter with 200-ps resolution. *IEEE Trans. Instrum. Meas.*, 46(4):851–856, Aug. 1997.

[27] S. C. Schneider R. H. Brown, and. Velocity observations from discrete position encoders. In *IEEE conf. of the Industrial Electronics Society*, pages 1111–1118, 1987.

[28] N. K. Boggarpu and R. C. Kavanagh. New learning algorithm for high-quality velocity measurement from low-cost optical encoders. In *IEEE Instrumentation and Measurement Technology Conference (I^2MTC 2008)*, pages 1908–1913, Victoria, Canada, May 2008.

[29] M. Prokin. Double buffered wide-range frequency measurement method for digital tachometers. *IEEE Trans. Instrum. and Meas.*, 40(3):606–610, Jun 1991.

[30] M. Prokin. Extremely wide-range speed measurement using a double-buffered method. *IEEE Trans. Instrum. and Meas.*, 41(5):550–559, Oct. 1994.

[31] M. Prokin. Speed measurement using the improved dma transfer method. *IEEE Trans. Ind. Electron.*, 38(6):476–483, Dec 1991.

[32] G. Bucci and C. Landi. Metrological characterization of a contactless smart thrust and speed sensor for linear induction motor testing. *IEEE Trans. Instrum. and Meas.*, 45(2):493–498, Apr 1996.

[33] Jian-Zhong Tang, Yujian Fan, Bin Fei, and Dan Chen. New method for dividing encoder signals by means of computer. *SPIE - Measurement Technology and Intelligent Instruments*, 2101(1):901–904, 1993.

[34] R. C. Kavanagh. Signal processing techniques for improved digital tachometry. In *IEEE Intl. Symp. Industrial Electronics (ISIE2000)*, pages 511–517, L'Aquila, Italy, Jul. 2002.

[35] E. Galván, A. Torralba, and L. G. Franquelo. A simple digital tachometer with high precision in a wide speed range. In *Proc IEEE IECON'94*, pages 920–923, Bologna, Italy, Sept. 1994.

[36] E. Galván, A. Torralba, and L. G. Franquelo. ASIC implementation of a digital tachometer with high precision in a wide speed range. *IEEE Trans. Instrum. Meas.*, 43(6):655–661, Dec. 1996.

[37] N. Chaudhuri, S. Ghosh, and A. M. Ghosh. Wide-range precision speed measurement with adaptive optimization using a microcomputer. *IEEE Trans. Ind. Electron.*, IE-30:369–373, Nov. 1983.

[38] P. Bhatti and B. Hannaford. Single-chip velocity measurement system for incremental optical encoders. *IEEE Trans. Control. Syst. Technol.*, 5(6):654–661, Nov. 1997.

[39] F. Thomas, J. K. Kishore, K. M. Bharadwaj, M. M. Nayak, and V. K. Agrawal. Design and implementation of a wheel speed measurement circuit using field-programmable gate array in a spacecraft. *Microprocessors and Microsystems*, 22(9):553–560, Mar. 1999.

[40] R. C. Kavanagh. An enhanced constant sample-time digital tachometer through oversampling. *Trans. Inst. Meas & Control*, 26:416–422, Apr. 2004.

[41] G. Kahl. Digital measurements of transient angular speeds with high resolution. In *Proc. Intl. Conf. Microelectronics in Power Electronics and Electrical Drives*, pages 69–73, Darmstadt, Germany, 1982.

[42] S. K. Kaul, R. Koul, C. L. Bhat, I. K. Kaul, and A. K. Tickoo. Use of a 'look-up' table improves the accuracy of a low-cost resolver-based absolute shaft encoder. *Measurement Science and Technology*, 8:329–331, 1997.

[43] D. Mancini, A. Auricchio, M. Brescia, E. Cascone, F. Cortecchia, P. Schipani, and G. Spirito. Encoder system design : Strategies for error compensation. In *SPIE - The International Society for Optical Engineering*, volume 3351, pages 380–386, New York, USA, May 1998.

[44] R. C. Kavanagh. Probabilistic learning technique for improved accuracy of sinusoidal encoders. *IEEE Trans. Ind. Electron.*, 48(3):673–681, Jun. 2001.

[45] C. Wang, G. Zhang, S. Guo, and J. Jiang. Auto correction of interpolation errors in optical encoders. In *Proc. SPIE*, volume 2718, pages 439–447, May 1996.

[46] I. Ogura, Y. Suzuki, and N. Hagiwara. A method of improving the accuracy of rotary encoders using a code compensation technique. *Trans. IEE Japan*, 144c(1):75–88, Jun. 1994.

[47] J. Doernberg, Hae-Seung Lee, and D. Hodges. Full-speed testing of A/D converters. *IEEE Jnl. Solid-Stage Circuits.*, SC-19(6):820–827, Dec. 1984.

[48] K. Hachiya and T. Ohmae. Digital speed control system for a motor using two speed detection methods of an incremental encoder. In *Power Electronics and Applications, 2007 European Conference on*, pages 1–10, Sept. 2007.

[49] K. K. Tan and K. Z. Tang. Adaptive online correction and interpolation of quadrature encoder signals using radial basis function. *IEEE Trans. Control. Syst. Technol.*, 13(3):370–377, May 2005.

[50] Agilent Technologies. Quick assembly two and three channel optical encoders, datasheet.

[51] Nemicon Rotary Encoders. OVW incremental rotary encoder datasheet. URL http://www.nemicon.com/pdf/OVW-2.pdf.

[52] Omron Industrial Automation. Omron incremental rotary encoder datasheet. URL http://www.ia.omron.com/data_pdf/e6c2-c_ds_csm493.pdf.

[53] S. Wolf. *Silicon Processing for the VLSI Era.* Lattice Press, California, U.S, 1990.

[54] J. Rose, A. El Gamal, and A. Sangiovanni-Vincentelli. Architecture of field-programmable gate arrays. *Proceedings of the IEEE*, 81(7):1013–1029, Jul. 1993.

[55] S. D. Brown, R. J. Francis, J. Rose, and Z. G. Vranesic. *Field-Programmable Gate Arrays*. Kluwer Acad. Publ., 1992.

[56] H.-C. Hsieh, W.S. Carter, J. Ja, E. Cheung, S. Schreifels, C. Erickson, P. Freidin, L. Tinkey, and R. Kanazawa. Third-generation architecture boosts speed and density of field-programmable gate arrays. In *Proc. IEEE Custom Integrated Circuits Conference, 1990*, pages 31.2/1–31.2/7, May 1990.

[57] Trimberger. S. Beyond logic - FPGAs for digital systems. In W.Moore and W.Luk, editors, *FPGAs*, pages 39–45. Oxford, Abbington EE and CS Books, 1991.

[58] Xilinx Inc. The programmable logic data book, 1996.

[59] D. Bursky. Streamlined RAM-based family of FPGAs trims system cost. In *Electronic design*, volume 46, pages 98–100, Penton Media, Cleveland, OH, USA, Feb 1998.

[60] dSPACE. DS1104 r & d controller board. URL http://www.dspaceinc.com/en/home/products/hw/singbord/ds1104.cfm.

[61] Reliance Electric Company Electro-Craft. *BRU-200/BRU-500 Brushless Drives Instruction Manual*. Reliance Electric Company, 1992.

[62] Aerotech Ltd. 1000 Series brush, rotary dc servomotors. URL http://www.aerotech.com/products/pdf/1000.pdf.

[63] O. Axelsson. *Iterative Solution Methods*. Cambridge University Press, New York, USA, 1st edition, 1994.

[64] A.C.Bajpai, I.M.Calus, and J.A.Fairley. *Numerical Methods for Engineers and Scientists*. John Wiley & Sons, Great Britain, 1st edition, 1975.

[65] P. Barooah and J.P. Hespanha. Distributed estimation from relative measurements in sensor networks. In *Third International Conference on Intelligent Sensing and Information Processing, 2005. ICISIP 2005.*, pages 226–231, Dec. 2005.

[66] Joo P. Hespanha Prabir Barooah, Neimar Machado da Silva. *Distributed Computing in Sensor Systems*, volume 4026 of *Lecture Notes in Computer Science*, chapter on Distributed Optimal Estimation from Relative Measurements for Localization and Time Synchronization, pages 266–281. Springer Berlin / Heidelberg, 2006.

[67] P. Barooah and J.P. Hespanha. Estimation from relative measurements: Electrical analogy and large graphs. *IEEE Trans. Sig. Proc.*, 56(6):2181–2193, Jun. 2008.

[68] V. Delouille, R. Neelamani, and R. Baraniuk. Robust distributed estimation in sensor networks using the embedded polygons algorithm. In *Information Processing in Sensor Networks, 2004. IPSN 2004. Third International Symposium on*, pages 405–413, Apr. 2004.

[69] The Mathworks Inc. Signal processing toolbox-for use with matlab, 1994.

[70] Richard O. Duda, Peter E. Hart, and David G. Stork. *Pattern Classification*. John Wiley & Sons, Inc, New York, United States of America, 2nd edition, 2001.

[71] R. C. Kavanagh. Performance analysis and compensation of M/T-type digital tachometers. *IEEE Trans. Instum. Meas.*, 50(4):965–970, Aug. 2001.

[72] W. Schumacher and G. Heinemann. Fully digital control of induction motor as servo drive. In *Proc. 1st. Euro. Conf. Power Electronics and Appls. (EPE'85)*, pages 2.191–2.196, Brussels, Belgium, Oct 1985.

[73] B. C. Kuo and T. Kubis. Prediction of self-sustained oscillations in digital systems with velocity decoders. In *Proc. 11th Annual Symp. Incremental Motion Control System and Devices*, pages 79–86, Jun. 1982.

[74] B. C. Kuo. Prediction of self-sustained oscillations in digital control systems by the discrete describing function. In *Proc. 11th Annual Symp. Incremental Motion Control System and Devices*, pages 65–78, Jun. 1982.

[75] P. J. Roche, J. M. D. Murphy, and M. G. Egan. Reduction of quantisation noise in position servosystems. In *Proc. IEEE Conf. Ind. Elect. and Control (IECON '92)*, pages 464–469, San Diego, U.S.A, Nov. 1992.

[76] K. Fujita and K. Sado. Instantaneous speed detection with parameter identification for ac servo systems. *IEEE Trans. Ind. Applicat.*, 28(4):864–872, July/August 1992.

[77] S. Sakai and Y. Hori. Ultra-low speed control of servomotor using low resolution rotary encoder. In *IEEE international conference on industrial electronics and control*, pages 615–620, 1995.

[78] N.K. Boggarpu and R.C. Kavanagh. New learning algorithm for high-quality velocity measurement and control when using low-cost optical encoders. *IEEE Trans. Instrum. Meas.*, 59(3):565–574, March 2010.