UNIVERSITY OF
KWAZULU-NATAL

INYUVESI
YAKWAZULU-NATALI

Open Architecture Control System for a Modular Reconfigurable Machine Tool

Submitted by:

Mr AQM Amra - 207505463

BSc Eng, UKZN

Supervisor:

Prof. G Bright

May 2013

Submitted in fulfilment of the academic requirements for the degree of Master of Science in Engineering at the School of Mechanical Engineering, University of KwaZulu-Natal.

Declarations

Declaration by supervisor:

As the candidate's Supervisor I agree to the submission of this dissertation:

Signed: ________________________ Date: ________________________

Prof. Glen Bright

Declaration by Author:

I, Abdul Qadir Amra, declare that:

(i) The research reported in this dissertation, except where otherwise indicated, is my original research.

(ii) This dissertation has not been submitted for any degree or examination at any other university.

(iii) This dissertation does not contain other persons' data, pictures, graphs or other information, unless specifically acknowledged as being sourced from other persons.

(iv) This dissertation does not contain other persons' writing, unless specifically acknowledged as being sourced from other researchers. Where other written sources have been quoted, then:

a) Their words have been re-written but the general information attributed to them has been referenced;

b) Where their exact words have been used, their writing has been placed inside quotation marks, and referenced.

(v) This dissertation/thesis does not contain text, graphics or tables copied and pasted from the Internet, unless specifically acknowledged, and the source being detailed in the dissertation/thesis and in the References sections.

Signed: ________________________ Date: ________________________

Mr. Abdul Qadir Amra

Declaration: Publications

Amra, A. Bright, G. Padayachee, J; "Open Architecture Controller for Modular Reconfigurable Machine Tool". In 20th International Conference on Mechatronics and Machine Vision in Practice (M2VIP); 20th September 2013; Ankara; Turkey.

Acknowledgements

In the name of *Allah*, the most beneficent, the most merciful. All praise and thanks is due to Allah, without him nothing would be possible. To *Allah* for the will power, mental and physical strength and support that he has blessed me with throughout this research.

To my wife Aalia, for her understanding and love, without whom I would not have been able to achieve this. For her motivation, support, encouragement, patience, assistance and sacrifices that she made in the past 2 years.

To my family for their continued support, motivation and help throughout this research.

To Professor Glen Bright, I thank him for the supervision, motivation and support throughout this research.

I would also like to thank the following people and departments:

- The staff at the School of Mechanical Engineering (UKZN) for their outstanding support.
- Mr. Jared Padayachee for his assistance and guidance.
- My colleagues at the Mechatronic and Robotic Research Group for their assistance.

Abstract

The present day manufacturing environment has forced manufacturing systems to be flexible and adaptable to be able to match the product demands and frequent introduction of new products and technologies. This research forms part of a greater research initiative that looks at the development of the reconfigurable manufacturing paradigm. Previous research has shown that the lack of development of a Modular Reconfigurable Manufacturing Tools (MRMT) and Open Architecture Control System (OACS) is currently a key limiting factor to the establishment of Reconfigurable Manufacturing Systems (RMS), which has been the primary motivation for this research.

Open Architecture (OA) systems aim to bring the ideas of RMS to control systems for machining systems. An OA system incorporates vendor neutrality, portability, extendibility, scalability and modularity. The research has proposed, designed and developed a novel solution that incorporates these core principles allowing the system to be flexible in mechanical and control architectures. In doing so, the system can be reconfigured at any time to match the specific manufacturing functionality required at that time thereby prolonging the lifecycle of the machine via multiple reconfigurations over time, in addition to decreasing the cost of system modifications due to a well-defined modular system. The reconfiguration and machining variance is achieved by the introduction of mechanical and control modules that extend the Degrees of Freedom (DOF's) available to the system.

The OACS has been developed as a modular solution that links closely to the existing mechanical modularity on the RMT to maximize the reconfigurability of the system. The aim was to create a one to one link between mechanical and electronic hardware and the software system. This has been achieved by the addition of microcontroller based distributed modules which acts as the interface between the electro-mechanical machine axes via hardwired signals and the host PC via the CAN bus communication interface.

The distributed modules have been developed on different microcontrollers, which have successfully demonstrated the openness and customizability of the system. On the host PC, the user is presented with a GUI that allows the user to configure the control system based on the MRMT physical configuration. The underlying software algorithms such as, text Interpretation, linear interpolation, PID or PI controllers and determination of kinematic viability are part of the OACS and are used at run time for machine operation.

The machining and control performance of the system is evaluated on the previously developed MRMT. The performance evaluation also covers the analysis of the reconfigurability and scalability of the system. The research is concluded with a presentation based on conclusions drawn from the research covering the challenges, limitations and problems that OA and RMS can face before MRMT become readily available for industry.

Table of Contents

Declarations ... i

Acknowledgements ... ii

Abstract ... iii

Table of Contents ... iv

List of Acronyms and Abbreviations ... viii

Nomenclature ... x

List of Figures ... xii

List of Tables ... xvi

1. Introduction ... 1
   - 1.1 Manufacturing System Environment ... 1
   - 1.2 Research Question ... 2
   - 1.3 Research Objectives ... 2
   - 1.4 MSc Research Contribution ... 2
   - 1.6 Outline of Dissertation ... 3
   - 1.7 Chapter Summary ... 4
2. Manufacturing Systems ... 5
   - 2.1 Dedicated Manufacturing Systems ... 5
   - 2.2 Flexible Manufacturing Systems ... 5
   - 2.3 Intelligent Manufacturing Systems ... 7
   - 2.4 Reconfigurable Manufacturing Systems ... 7
   - 2.5 Manufacturing System Life Cycles ... 11
   - 2.6 Chapter Summary ... 12
3. Open Architecture, Modular and Distributed Control Systems ... 13
   - 3.1. Essential Characteristics of Open Architecture Controllers ... 13
   - 3.2. Emerson DeltaV ... 16
   - 3.3. Siemens SIMATIC ... 18
   - 3.5. Chapter Summary ... 19
4. Mechanical Systems ... 20
   - 4.1 Computer Numerically Controlled Machines ... 20
   - 4.2. Modular Reconfigurable Manufacturing Tool ... 21
   - 4.4. Chapter Summary ... 27
5. Open Architecture System for a Modular Reconfigurable Machine Tool ... 28
   - 5.1 Research and Design Outlines ... 28
   - 5.2 Mechatronic Design Approach ... 28
   - 5.2 OACS High Level Design ... 29

5.3 Distributed Modules 32
5.4. Buffer Layers 34
5.5. Chapter Summary 35
6. Electronic Subsystems 36
6.1 Overview 36
6.2 Host PC 36
6.3 Distributed Modules 37
6.3 Multiple Microcontroller Implementation 38
6.4 Communications Network 40
6.5 CAN bus 42
6.6 CAN bus Hardware 43
6.7 Message Packet Formatting 45
6.8 Spindle and Axis Speed Control 47
6.9 Motor Noise Cancellation 48
6.10 Collision Detection 49
6.11 Position Feedback Encoders 51
6.12 Vibration Sensor: Accelerometer 52
6.13 Power Distribution Network 53
6.14. Chapter Summary 54
7. Control Algorithms for Open Architecture Control System 55
7.1. Kinematic Modelling 56
7.2. Control Theory 59
7.2.1 PID Controller 61
7.3. Program Interpretation and Validation 65
7.4. Interpolation 69
7.4.1. Linear Interpolation 69
7.4.2. Circular Interpolation 71
7.5. Acceleration and Deceleration Control 74
7.6. Encoder Position Algorithms 76
7.6. Chapter Summary 77
8. Open Architecture Control System Software 78
8.1 Overview 78
8.2 Development Environment 79
8.3 User Interface 79
8.4 Set-Up and Configuration 81
8.5 Data Downloading and Module Information 84
8.6 Contoller Selection 86
8.7 User Programming 87

8.8 Motor Control, Debugging and Performance Evaluation ... 88
8.9 Algorithm Editing and Tuning ... 89
8.10. Distributed Module Software ... 90
8.11 Chapter Summary ... 91
9. System Testing and Performance ... 93
9.1. MRMT Setup ... 93
9.2. Reconfiguration Times ... 95
9.3. PC Load ... 96
9.4. Acceleration and Deceleration Control Example ... 96
9.5. Accuracy and Repeatability ... 97
9.6. Vibrations ... 102
9.7. Power Distribution Network Loading ... 105
9.8. Response Times ... 107
9.9. Chapter Summary ... 109
10. Discussion ... 110
10.1 Performance of Electronic Subsystems ... 110
10.1.1 Position Control: Accuracy and Repeatability ... 110
10.1.2. Axes Speeds and Time to Execute Commands ... 110
10.1.3 Power Usage ... 111
10.1.4 Vibrations ... 112
10.1.5 Distributed Module Response Times ... 112
10.2 Performance of OACS on PC ... 113
10.2.1 Reconfiguration Times ... 113
10.2.2 CPU Load ... 113
10.3 OACS Integration with Mechanical System ... 113
10.4 OACS and OA standards ... 115
10.5 OACS on MRM for RMS ... 117
10.6 Challenges and Limitations for OA Systems ... 118
10.7 Chapter Summary ... 120
11. Conclusion ... 121
References ... 123
Appendices ... 126
Appendix A: Kinematic Modelling Data ... 126
X Axis - Base Module ... 126
Z Axis - Column Module ... 126
A Axis – Rotary Module ... 127

Drill Module 127

Modular Range Extension Arm 128

Appendix B: Distributed Module Schematics 129

Appendix C: PID Code 133

Appendix D: Sample Code for Text Interpretation 135

Appendix E: Linear Interpolation Code 139

Appendix F: Circular Interpolation Code 143

Appendix G: Acceleration Deceleration Code 146

Appendix H: Sample Code of CANbus Class 149

Appendix I: Sample Code for Generic Servo Module Class 154

Appendix J: Sample Code for Physical Configuration Tab 157

Appendix K: Sample Code for Data Download Tab 163

Appendix L: Sample Code for Motor Control Tab 166

Appendix M: Reconfiguration Time Data 168

Appendix N: Acceleration/Deceleration Control Example Data 169

Appendix O: Encoder Count Data 170

Time Comparisons Data 171

Appendix P: Vibration Intensity Recording Data 172

Appendix Q: Current Consumption Test Data 178

Note: Only samples of the C# code for the OACS GUI and the C++ code for the distributed modules has been included in the appendices due to the length of the code developed and written. The complete code that was written for the system is included on the attached CD.

List of Acronyms and Abbreviations

ADC	Analog to Digital Convertor
API	Application Programming Interface
CAD	Computer Aided Design
CAM	Computer Aided Manufacturing
CAN	Controller Area Network
CAPP	Computer Aided Part Planning
CIM	Computer Integrated Manufacturing
CNC	Computer Numerically Controlled
COTS	Commercially Off the Shelf
DC	Direct Current
DCS	Distributed Control System(s)
DMS	Dedicated Manufacturing System(s)
DOF	Degrees of Freedom
EI	Electromagnetic Interference
FIFO	First In First Out
FMS	Flexible Manufacturing System(s)
GUI	Graphical User Interface
HMI	Human Machine Interface
HTM	Homogenous Transformation Matrix
IEEE	Institute of Electrical and Electronics Engineers
IMS	Intelligent Manufacturing System(s)
IS	Intrinsically Safe
LMS	Lean Manufacturing System(s)
MRMT	Modular Reconfigurable Manufacturing Tool(s)
NC	Numeric Control
OA	Open Architecture
OACS	Open Architecture Control System(s)
OMAC	Organisation for Machine Automation and Control
PC	Personal Computer
PI	Proportional Integral
PID	Proportional Integral Derivative
PLC	Programmable Logic Controller(s)
PWM	Pulse Width Modulation
RMS	Reconfigurable Manufacturing System(s)
RMT	Reconfigurable Manufacturing Tool(s)
USB	Universal Serial Bus

VN	Vendor Neutral
VS	Vendor Specific

Nomenclature

q_i	Joint Variable
θ_i	Angle Joint Variable
d_i	Displacement Joint Variable
M	Homogenous Transformation Matrix
T_j^i	HTM that expresses the overall configuration of the MRMT
R_n^0	3x3 rotation motion matrix
O_n^0	3x1 translational motion matrix
u	PID controller output
y	Process output
r	Reference position
e	Error
K	Electromotive constant
J	Moment of inertia of the mechanical system about the axis of the motor
b	Damping coefficient of the mechanical system on the axis
L	Motor inductance
R	Electrical resistance of the motor
$P(s)$	Servo Motor Transfer Function
$G_c(s)$	PID Transfer Function Continuous domain
$G(z)$	PID Transfer Function discrete time domain
K_p	Proportional gain
K_i	Integral gain
K_d	Derivative gain
T	Sample Time
L	Delay time
T	Time constant
F	Feed rate
L	Interpolation Length
t_{ipo}	Interpolation time
X_i	Displacement at i
x_{i+1}	Displacement at $i+1$
N	Number of Interpolation iterations
$V_x(t)$	X axis velocity function
$V_y(t)$	Y axis velocity function
ER	Radial error
$x[n]$	Interpolation output and input signal for filter
$h[n]$	Input to a filter with impulse response

$y[n]$	Convolution output
τ	Discrete time
A_i	Input data shifter at the i^{th}iteration
ΔX_0	Acceleration and deceleration output pulses
V_{ref}	Reference voltage
V_{norm}	Supply voltage to the accelerometer
n	Number of the bits on the ADC

List of Figures

Figure 1: Manufacturing System Evolution from 1980-2000 1

Figure 2: Manufacturing Systems Paradigms 8

Figure 3: Illustration of the Reconfiguration of a RMS 12

Figure 4: Criteria for Open Control Systems 14

Figure 5: Three Critical Aspects of OA Controllers- Add, Swap and Integrate 15

Figure 6: OSACA Reference Model 16

Figure 7: (a) DeltaV System Architecture (b) Plug and Play Modular Cards 17

Figure 8: CNC System Architecture 20

Figure 9: CNC Servo Driving Mechanism 21

Figure 10: MRMT Conceptual Module Library 22

Figure 11: Machine Reconfiguration 23

Figure 12: Drawing Indication: Two Bolted Interfaces on Mechanical Module 23

Figure 13 (a) Coordinate System (b) Absolute Position Reference 24

Figure 14: X Axis Drive Mechanism 25

Figure 15: Z Axis Drive Mechanism 25

Figure 16: A Axis Drive Mechanism 26

Figure 17: Drill Head Module 26

Figure 18: Accessory Module - Modular Range Extension Arm 26

Figure 19: Hardware/Software Co-Design Approach 29

Figure 20: Integration of Distributed Modules 30

Figure 21: High Level System Design 31

Figure 22: Examples of the Software Modules In System Memory 32

Figure 23: Example of the Components of a Distributed Controller 33

Figure 24: Module Linking With Buffers 34

Figure 25: OACS Architecture 36

Figure 26: FEZ Panda 2 Board 37

Figure 27: Diagram Illustrating the Tasks of the Distributed Spindle Module 38

Figure 28: Diagram Illustrating the Tasks of the Distributed Servo Module 38

Figure 29: chipKIT Max32 39

Figure 30: Arduino UNO 39

Figure 31: Comparison Between the Wiring for a Multi-wire vs CAN bus Network 41
Figure 32: Pin Out Configuration of a 9 pole SUB-D Connecter 43
Figure 33: Peak Systems PCAN PCI Card 44
Figure 34: Illustration of CAN bus Network and Node Connections 44
Figure 35: chipKIT Network Shield 45
Figure 36: CAN-BUS Shield 45
Figure 37: CAN bus Message Fields 46
Figure 38: (a) Pin Out Configuration of Motor Driver (b) Assembled Motor driver 47
Figure 39: Motor Terminals With Added Capacitors to Filter Out Noise 49
Figure 40: Limit Switch 49
Figure 41: Illustration of Limit Switches on the MRMT 50
Figure 42: Schmitt Trigger Connection Diagram for the 555 Timer 51
Figure 43: (a) Block Diagram for the Optical Encoder (b) Output Signals From the Encoder 52
Figure 44: SparkFun ADXL335 Breakout Board Triple Axis Accelerometer 53
Figure 45: Power Distribution Network 54
Figure 46: Flowchart Illustrating Embedded Algorithms 56
Figure 47: Coordinates References Between Movements 58
Figure 48: System Architecture Highlighting Control Functionality 60
Figure 49: Closed Loop Feedback Control System 61
Figure 50: PID Block Diagram 61
Figure 51: Controller Response 64
Figure 52: Ziegler Nichols Tuning 65
Figure 53: Text Interpretation and Validation Flowchart 68
Figure 54: (a) 2 Axis Input Parameters (b) Interpolated results 70
Figure 55: 2 Axis Linear Interpolation Example 70
Figure 56: Reference Word Algorithm 71
Figure 57: Reference Word Algorithm for Circular Interpolation 72
Figure 58: Radial and Cord Height Errors for Circular Interpolation 73
Figure 59: Convolution Example 74
Figure 60: Input and Output Pulse Train Profiles 75
Figure 61: Illustration for S-Shape Acceleration/Deceleration Control [28] 75

Figure 62: Output Signals from the Encoder........ 76
Figure 63: System Architecture........ 79
Figure 64: Graphical User Interface........ 80
Figure 65: Flowchart of Sequence of Events........ 81
Figure 66: GUI Showing Results of a Bus Scan........ 82
Figure 67: GUI Showing Numbering Sequence........ 83
Figure 68: Illustration of the Assigning of Classes........ 84
Figure 69: GUI Data Downloads........ 85
Figure 70: GUI Module Information........ 86
Figure 71: GUI Controller Selection........ 86
Figure 72: Program Editor........ 87
Figure 73: Program Editor with User Entered Program........ 88
Figure 74: GUI Motor Control and Debugging........ 89
Figure 75: GUI Algorithm Editor........ 90
Figure 76: Microcontroller Software Flowchart........ 92
Figure 77: MRMT Configuration for Testing........ 93
Figure 78: Distributed Module - Arduino UNO (Linear Axis Z)........ 94
Figure 79: Distributed Module – chipKIT (Linear Axis X)........ 94
Figure 80: Distributed Module – FEZ Panda 2 (Rotary Axis A)........ 94
Figure 81: Distributed Module – FEZ Panda 2 (Spindle Module)........ 95
Figure 82: Graph of Software Reconfiguration Time........ 95
Figure 83: CPU Load During OACS Operation........ 96
Figure 84: Graph of Acc/Dec Output Pulse........ 97
Figure 85: Graph of Acc/Dec Output Pulse........ 97
Figure 86: X Axis Encoder Counts........ 98
Figure 87: Z Axis Encoder Counts........ 99
Figure 88: A Axis Encoder Counts........ 99
Figure 89: A Axis Encoder Counts (Lower Encoder Resolution)........ 101
Figure 90: Axis Movement vs Average Time........ 102
Figure 91: Recorded Vibrations Due to Axis Movement and Drill Operation........ 103
Figure 92: Recorded Vibrations from Drilling 20mm into Product........ 104

Figure 93: Recorded Vibrations from Drilling 40mm into Product 104
Figure 94: Graph of X Axis Current Consumption 105
Figure 95: Graph of Z Axis Current Consumption 105
Figure 96: Graph of A Axis Current Consumption 106
Figure 97: Graph of Unloaded Drill Current Consumption 106
Figure 98: Graph Drill Current Consumption During Operation 107
Figure 99: Screenshot of Data from CAN bus Monitor 108
Figure 100: MRMT and OACS Configuration 2 Modules 114
Figure 101: MRMT and OACS Configuration 4 Modules 115
Figure A102: X Axis with i and i+1 Reference Points 126
Figure A103:Z Axis with i and i+1 Reference Points 126
Figure A104: (a) A Axis with i (b) i+1 Reference Points 127
Figure A105: Drill Module i and i+1 Reference Points 127
Figure A106: Modular Range Extension Arm (a) i and (b) i+1 Reference Points 128
Figure B107: Board ID 3 Schematic 129
Figure B108: Board ID 2 Schematic 130
Figure B109: Board ID 4 Schematic 131
Figure B110: Board ID 5 Schematic 132

List of Tables

Table 1: Comparison of DMS, FMS and IMS 7

Table 2: Comparison Between DMS, RMS and FMS [9] 10

Table 3: Comparison of Critical Features of the 3 Boards 40

Table 4: Comparison Between Profibus, Foundation Field Bus, Modbus and CAN bus 41

Table 5: Unique Node Addresses/Identifiers 43

Table 6: Table Showing Packet Message Formats for Transmissions from Host PC 46

Table 7: Table Showing Packet Message Formats for Transmissions from Distributed Modules 47

Table 8: Control Signals and Operation for Motor Drivers 48

Table 9: Schmitt Trigger Detector Input Conditions and Outputs 51

Table 10: Trial and Error Tuning Guidelines 64

Table 11: Ziegler Nichols Tuning Rules 65

Table 12: Reduced NC command Instruction Set 66

Table 13: Logic for Decoding Quadrature Encoder Waveforms 76

Table 14: Summary of MRMT Design Specifications 97

Table 15: Summary of Test Results for Repeatability 99

Table 16: Summary of Accuracy Test Results for Translational Axes 100

Table 17: Summary of Test Results for Accuracy and Repeatability for A Axis 101

Table 18: Comparison of Axis Speeds 102

Table 19: Range of Accelerometer Readings Captured 103

Table 20: Summary of Distributed Module Response Times 109

Table 21: Summary of OACS Core Principles 115

Table 22: Comparison of OACS and Other Research Attempts 116

Table A23: X Axis Design Data 126

Table A24: Z Axis Design Data 126

Table A25: A Axis Design Data 127

Table A26: Drill Module Design Data 127

Table A27: Modular Range Extension Arm Design Data 128

Table M28: Reconfiguration Times for 3 Modules (Familiar User) 168

Table M29: Reconfiguration Time for 3 Modules (New User) 168

Table N30: Acceleration/Deceleration Results (Constant Multiplier Values) 169

Table N31: Acceleration/Deceleration Results (Varied Multiplier Values) 169
Table O32: Encoder Count - X Axis (80mm Movement) 170
Table O33: Encoder Count - Z Axis (50mm Movement) 170
Table O34: Encoder Count - A Axis (345° Movement) 170
Table O35: Time (sec) Comparisons for Movement on All Axes 171
Table P36: Vibration Intensity Due to Axis Movement and Drill Operation 172
Table P37: Vibration Intensity During Drilling Operation – Drill In/Out 20mm 172
Table P38: Vibration Intensity During Drilling Operation – Drill In/Out 40mm 174
Table Q39: Current Consumption (A) - A Axis 178
Table Q40: Current Consumption (A) - X Axis 178
Table Q41: Current Consumption (A) - Z Axis 179
Table Q42: Current Consumption(A) Unloaded Drill 180
Table Q43: Current Consumption(A) During Drilling Operation 180

1. Introduction

Chapter 1 introduces the reader to the scope of the research. A summary of the challenges and current state of present day manufacturing systems is presented. This discussion identifies the limitations of RMS, and this has set the basis for the motivation for the research. Thereafter the aims and objectives for the research are presented followed by a short summary of the remaining chapters in the dissertation.

1.1 Manufacturing System Environment

The demands on manufacturing systems have changed significantly over the past few decades. Early in the 20th century, mass production was required to meet consumer demands and market requirements [1]. This demand for mass production led to the research and development of machines and systems that were capable of rapidly producing large quantities of products at lower costs. However later in the 20th century, the market demands changed again, and a trend was seen that the time to market for products was decreasing and the complexity of product machining requirements was increasing [2].

In the last decade, manufacturing demands have changed based on the idea of mass customization. In a workplace where globalization has come to the fore recently, global economic competition has forced manufacturers to re-evaluate how products are made and the cost effectiveness of production. Figure 1 [2] shows a timeline of events regarding the evolution of manufacturing systems, illustrating a decrease in the time to market and an increased level of product complexity from 1980 to 2000.

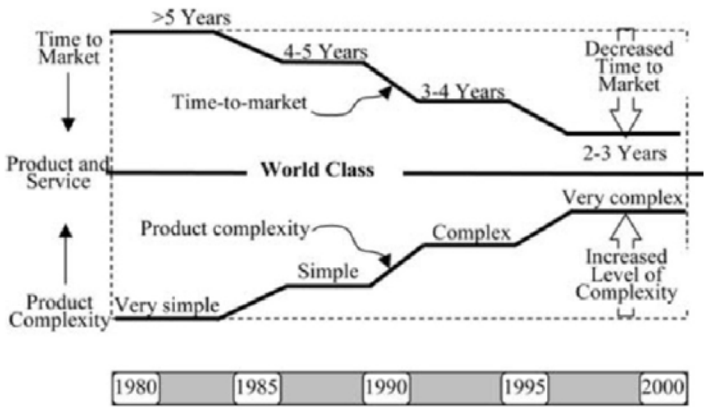


Figure 1: Manufacturing System Evolution from 1980-2000

In summary, the dynamic market has meant that a manufacturing system needs to manage the:

- Frequent introduction of new products;
- Fluctuations in consumer demands;
- A greater demand for product customization;
- Regular introduction of new technologies and legislations for production.

Reconfigurable Manufacturing Systems (RMS) have emerged in recent years with the aim of managing the aforementioned points. From the conception of the ideas to the physical implementation, RMS is aimed at being inherently flexible in both software and hardware. This flexibility ensures that the system is able to keep up with the constantly growing, evolving and varying production requirements, while still maintaining a high throughput thereby to prolonging the beneficial operation period of a machine.

The flexible nature of RMS requires equally flexible control systems that can quickly and reliably adjust its control functionality depending on the systems available hardware modules at any given time. In the surveys conducted by Mehrabi et al. [3], software issues posed the area of greatest concern for the successful development and implementation of RMS. In addition ElMaraghy has highlighted that key technologies such as modular machines and Open Architecture Control Systems (OACS) need further development for the realization of RMS [4].

1.2 Research Question

Reconfigurable Machine Tools (RMT) are a developing technology for advanced manufacturing systems of the future. The flexible and transformable nature of RMT requires equally flexible and reconfigurable control systems. According to Katz [5] these control systems, *"are designed to allow changes in machine configuration according to changes in production requirements."* This flexibility ensures that the machines, in hardware and software architectures can match changes, production requirements and evolving manufacturing objectives.

The research has involved the research, design and development of a Reconfigurable Open Architecture Control System for a RMT while attempting to answer whether an OACS can enable a RMT to assist advanced manufacturing systems in efficiently responding to frequent product and market changes.

1.3 Research Objectives

The objectives of this MSc research were as follows:

- To conduct a literature review of manufacturing engineering, manufacturing technology, open architecture systems and reconfigurable machine tools.
- To define the key characteristics that are required in the development of an open architecture control system.
- The research, design and development of electronic subsystems on an RMT, namely:
    - Embedded controller platform;
    - Distributed control drives;
    - Communications network/interface.
- The research, design and development of an control system consisting of:
    - PC Based open architecture control application that consists of:
        - Application Programming Interfaces (API) for user and machine control functions;
        - External interfaces – Programming and communication interfaces;
        - Internal interface – Control and interaction of mechanical modules.
- Analysis and validation of the performance of the system against predefined specifications.

1.4 MSc Research Contribution

The research covers the design and development of an OACS for a Modular Reconfigurable Machine Tool (MRMT). The MRMT was designed and developed as part of a previous research by

Padayachee et al. [6]. The MRMT is designed from the base up to the tool head to be modular, thereby allowing the MRMT to be reconfigurable, flexible, transformable, and scalable thus satisfying the RMS criteria. However, just as the modular mechanical hardware builds up the MRMT allowing it to change its functionality, in order to fully realise the benefits of MRMT, an electronic and control software system which can exhibit the same characteristics is necessary.

The research has proposed the use of a novel solution of a distributed microcontroller-based controller network linked together with a communications interface to create a modular distributed electronic software system. This modular approach supports the aim of a plug and play system of mechanical, electronic and software components. The bus network topology allows and assists with the seamless integration of new modules when the user reconfigures the MRMT. The microcontroller based distributed electronic modules performs the tasks of linking motor drivers, axis control, and collecting data from the sensory circuits.

In order for the electronic hardware systems to be reconfigurable, a modular OACS is required; the software system just like the hardware systems needs to be modular to allow for reconfigurability, flexibility, transformability, and scalability. To achieve this modularity, the software system is designed using C++, an object orientated class based programming language. This approach uses the class functionality of C++ to create a one to one link between the distributed electronic network and the software modules, in its aim to achieve a modular software system that links to the physical distributed modules.

The software system is presented with a tabbed Graphical User Interface (GUI) allowing the end user options for set up, configuration, tuning, debugging and monitoring of the MRMT. All of the aforementioned are implemented to satisfy the aims of an OACS. Invisible to the user is the underlying communication and control routines. User-specific programs can be programmed in and a text interpretation routine is run to validate the user programs.

Thereafter the movements are passed through interpolation, PID control and Acceleration and Deceleration control routines which determine the tool path profiles and the commands for each axis. Finally, these commands are passed on to distributed modules which interpret and carry out the correct commands. Feedback from the sensors is processed and fed back to the control routines to monitor and correct the machine movements.

The research has designed and implemented an OACS that is based on a GUI which runs of a standard PC which achieves the machine control and interfacing to the distributed modules. The research has also designed and implemented the distributed modules, the communications interface and associated sensory and interfacing modules. The OACS developed has been tested on an existing MRMT, and results were compared to a set of pre-defined performance benchmarks.

1.6 Outline of Dissertation

Chapter One: Presents the reader with the background of the Research, the motivation for the research, and a summary of what the research has entailed.

Chapter Two: Consists of a literature review analysing the existing manufacturing systems, their successes and failures which led to the motivation and development of the RMS paradigm.

Chapter Three: Consists of a literature review and analysis of the essential characterises of open architecture systems, industrially available Distributed and Modular Control Systems which have influenced the OACS design and implementation.

Chapter Four: Presents the core ideas of CNC machining systems, followed by the motivation for Modular Machines. Design and engineering details of the Modular Machine that the OACS has been tested on is also presented.

Chapter Five: Here the concept of an OACS and the design proposal for the OACS that has been developed is presented based on motivations from the literature review and analysis of existing controls systems.

Chapter Six: A core aspect of the design, which supports the idea of an OACS is the electronic subsystems and distrusted modules. The engineering, design and implementation of the electronic subsystems are presented as well.

Chapter Seven: Presents the various control and interpretation algorithms that the OACS uses from start-up, to configuration to machining.

Chapter Eight: The GUI sits over the various control algorithms presented in Chapter 7 and herein PC based OACS covering the software implementation of the GUI is presented. The various functions of the GUI are covered showing the reader an overview of the OACS.

Chapter Nine: Presents the MRMT and OACS setup that is used to test the OACS followed by test results and a short discussion based on the testing of the OACS on a MRMT.

Chapter Ten: Discussion evaluating the performance of the system based on the test results. The successes and failures of the OACS is also evaluated in light of RMS. Followed by a discussion on the challenges, limitations and further research.

Chapter Eleven: Concludes the research dissertation based on the principles of OACS and the aims set out in Chapter 1.

1.7 Chapter Summary

Chapter 1 introduces the reader to the scope of the research. A summary of the challenges and current state of present day manufacturing systems has been covered, where the key challenges facing manufacturing systems have been discussed. This discussion identifies the limitations of RMS, and this has set the basis for the motivation for the research. Furthermore the aims and objectives for the research have been presented followed by a short summary of the remaining chapters in the dissertation.

2. Manufacturing Systems

Chapter 2 is a presentation of a literature review analysing existing manufacturing paradigms, namely the DMS, FMS and IMS. Due to the inability of these systems to meet the current manufacturing demands, the RMS paradigm has emerged. RMS is expected combine the benefits of DMS, FMS and IMS to achieve a system that can constantly adapt its functionality through a reconfiguration process of both hardware and software.

2.1 Dedicated Manufacturing Systems

At the beginning of the 20th century, mass production was required to meet the demands of consumers [1]. This demand for mass production led to the research and development of machines and systems that were capable of rapidly producing large quantities of products at lower costs. A Dedicated Manufacturing System (DMS) is defined as *"A machining system designed for production of a specific part type at high volume. Cost-effectiveness is the driver achieved through pre-planning and optimization."* [4]

DMS are generally set up to manufacture a single product that will remain unchanged over the life cycle of the product, the system comprises of machines and tools that all are hardwired and hard-coded to carry out certain functions. DMS proved successful in delivering large volumes of products and even better results were found when the products had long life cycles.

In the 1980's there was a shift in the customer demands towards mass customization of products [4]. However DMS systems, *"are not designed such that they may be cost-effectively converted, the redesign and ramp-up of a modified (or entirely new) DMT will often be too costly."* [7] Therefore due to the fixed nature of DMS, although they are robust and relatively inexpensive at initial capital investment, they are not able to undergo upgrades of functionality changes thereby rendering DMS impractical in the modern day manufacturing environment.

As a result, the need arose for a better manufacturing system that would be more flexible as well as possess the ability to produce customized products. Lean Manufacturing Systems (LMS), which are considered to be an improvement of DMS for mass production has a small degree of inherent flexibility. This flexibility was centred on the technology selection and the automated equipment used to build up the LMS. The flexibility allows configurations machining changes around the part family.

LMS, due to this degree of flexibility, has the ability to modify product design unlike DMS but similar to DMS though, LMS have a production model which has optimum production efficiency when the market has stabilised [8]. Thus DMS and LMS both have a high efficiency and throughput of products, but with the limitation of a low responsiveness to market changes.

2.2 Flexible Manufacturing Systems

A Flexible Manufacturing System (FMS), *"is an integrated system of machine modules and material handling equipment under computer control for the automatic random processing of palletized parts."* [4]

FMS are a combination of Computer Numerically Controlled (CNC) machines, robots and other programmable automation electronics that is fixed in hardware but with software that can be programmed to vary the functionality. Flexible manufacturing aims to bring together the key advantages of dedicated and fixed manufacturing systems with programmed automated systems [8].

FMS aims to be selectively flexible in its software architectures in order to reduce the cost for production of several types of parts that can evolve over a period of time while maintaining a high degree of flexibility throughout the machine lifecycle. The idea is to maintain a minimum changeover cost and minimum changeover time of the system when a new part of the same family needs to be manufactured.

One of the aims of FMS was to increase its flexibility, thus allowing the system to be used to produce products with a higher variety as compared to DMS. This higher variety of products comes at a cost of lower throughput thereby limiting the success of FMS in the global market.

Due to the fixed mechanical architecture of the FMS, the system is limited in terms of customization, add-ons, upgrades and production capacity modifications. In addition, the fixed nature of the system, with the hope of being flexible to accommodate for changes in product requirements, means that the system is often installed with excessive functionality and more machines that are required at the time of installation. For example, "*a five-axis CNC may use only two axes in a given operation or only 6 tools of a 24-tool magazine may be utilized*" [7]. As a result, a FMS often as a significant initial capital investment.

In a survey by Mehrabi et al. [3], over 60% of FMS that are installed in industry are installed with more capacity than is required in order to accommodate for future expansions, whereas 55% of FMS that were installed with more features than required, did not utilize this excess capacity.

The strength of the FMS lies in its ability to manufacture and produce a range of products in a part family. Due to the range of functionality built in apriori, it has the ability to be in use for a greater period of time in the constantly changing markets of today. Unfortunately though the initial capital investment for a system with such functionality and capacity is extremely high and when compared to a DMS seems excessive and impractical.

Mehrabi et al. [3], established the following key points about FMS in industry:

- FMS not living up to its full potential;
- System often purchased with additional capacity for expansion, as a result a large initial investment was required;
- Issues such as: training, reconfigurability, reliability, maintenance and software were hindering the success of FMS.

Furthermore, the costs involved with modifying the system to change its functionality in hope of increasing its life cycle are significantly high. Thus it is evident that although FMS's are common in industry, they have many drawbacks and due to the initial capital investments required and dormant functionality that is not used, FMS are not practical.

The impracticality of FMS further leads to a desire for a new technology and system, which builds on the flexibility of a FMS system, but reduces the capital investment required for upgrades due to the constantly evolving manufacturing environment.

2.3 Intelligent Manufacturing Systems

The idea of Intelligent Manufacturing Systems (IMS) has come to the fore as a result of the global market demands on manufacturing systems. Setlak and Pieczonka [9] state that IMS is group of principles, methods and computer aided tools, *"equipped with artificial Intelligence tools and supporting designing, planning and manufacturing."*

IMS aim to use Computer Integrated Manufacturing (CIM), Computer Aided Design (CAD) and Computer Aided Manufacturing (CAM) which gives the system the ability to learn, obtain knowledge, adapt to a dynamic environment and to adapt to the arrangement of the IMS components [10].

The main idea behind IMS is the system reconfigurability and intelligence, where intelligence is defined as the ability to learn based on machine actions and monitoring of the success of operations, and reconfigurability is the capability of changing modules to achieve new functionality. IMS offer ultra-precision motion control, high speed manufacturing and high quality finishing at lower costs with better reliability [11].

Setlak and Pieczonka [9] highlight that IMS has had limited success primarily due to the problems in achieving a unified standard for IMS and lack of development of software architectures. Therefore any new IMS must be created with an OA system with a modular structure that allows for evolution as well as a distributed modular architecture to allow the system to be adaptable. Table 1 summarises the key features of the three manufacturing systems paradigms.

Table 1: Comparison of DMS, FMS and IMS

	DMS	FMS	IMS
Initial Capital Investment	Dependent on machining required for product	High	High
Expansion Capacity	None	Limited	High
Manufacturing of New Products	No	Same part family	Yes
Flexibility	None	Selected	Yes
Throughput	High	Medium	Medium
Reconfigurability	N/A	Limited	Yes

2.4 Reconfigurable Manufacturing Systems

A RMS is a system that incorporates the advantages of DMS and FMS by, *"designing it at the outset for rapid change in structure, as well as in its machines and controls, in order to quickly adjust production capacity and functionality in response to market or product changes."* [12]

A RMS aims to combine the advantages of the FMS and the DMS to produce a system that achieves a high throughput in addition to the necessary flexibility, which allows the system to evolve and change its production functionality efficiently and quickly when required to do so. A RMS system can be seen as a combination of both the FMS and DMS systems, attempting to bridge the gaps between the two and successfully produce large volumes of products while maintaining the flexibility required to produce a variety of products as seen in Figure 2 [13].

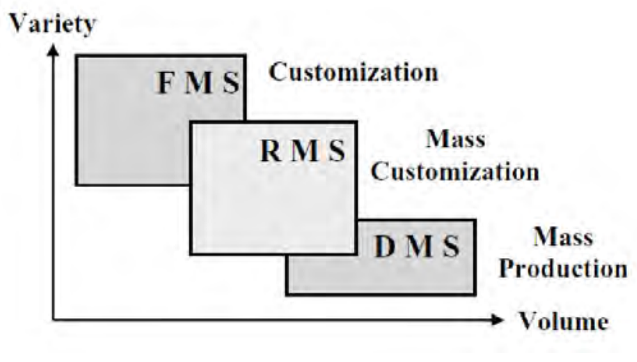


Figure 2: Manufacturing Systems Paradigms

The development of RMS was motivated for, primarily by the lack of alternate manufacturing strategies such as DMS and FMS to manage the:

- Frequent introduction of new products;
- Fluctuations in consumer demand;
- Greater demand for product customization;
- Regular introduction of new technologies and legislations for production.

RMS have emerged in recent years for the application in advanced manufacturing environments. From the conception of the ideas to the physical implementation of the RMS, the system is aimed at being inherently flexible in both software and hardware. According to Katz RMS, *"are designed to allow changes in machine configuration according to changes in production requirements."* [5]

This flexibility ensures that the system is able to keep up with the constantly growing, evolving and varying production requirements, while still maintaining a high throughput in addition to prolonging the beneficial operation period of a machine.

Furthermore ElMaraghy [4] notes that the RMS paradigm focuses on:

- Reducing the time for setting up new systems;
- Reducing the time for reconfiguring current systems;
- Reducing the down time of systems for the addition/integration of new modules (hardware and software) to modify the manufacturing process and add new functionality;
- Reducing the initial cost or capital investment of a system.

Unlike conventional DMS, FMS and CNC machines which are designed on a fixed hardware structure and closed software system allowing only limited access to the user, RMS adopts a similar approach to IMS, where the systems are designed to be reconfigurable in all aspects of hardware and software. In order to achieve this reconfigurability, the various parts of the hardware sub-systems are designed as separate modules which can be integrated together in a number of configurations to provide varying functionality.

Similarly, the control systems for the RMS and IMS are required to be modular in nature and its control functionality should vary depending on the available hardware modules. This modular concept is critical to the reconfigurability of the system and ensures that the system capacity and functionality is not fixed and predetermined. Rather these system characteristics allow for variability at any time thus contributing to the flexibility of the RMS [12]. In addition to this, the modular nature of these systems allows the system to be flexible in both the production of parts and in the structure of the system.

In order to achieve a high level of reconfigurability in the production environment, the design of RMS needs to address the following critical characteristics [14]:

- Modularity
  - The hardware and software components of the machine are all created in modules with well-defined interfaces such that a number of these modules may be integrated to produce a desired functionality.
- Convertibility
  - This refers to the ability to change/transform/convert the production functionality of the system depending on the production requirements.
- Scalability
  - This is the ability to add, integrate, remove or swap in new modules or machines thereby scaling the system up or down in terms of throughput and functionality with modules.
- Integrability
  - This refers to the ability to add, integrate or swap in new modules quickly, and precisely through a set of well-defined internal interfaces.
- Customization
  - This is the ability to adapt the functionality of the system to meet the needs of a specific manufacturer.
- Diagnosability
  - This refers to the ability to self-diagnose the state of the system in order to identify the causes of production defects and reliability issues.

Ensuring that these key characteristics are deeply embedded within the design philosophy and are addressed from early in the design phases ensures that the system has a high level of reconfigurability and low reconfiguration times. Furthermore there are also economic benefits that are seen with the implementation of RMS mainly from the increased life cycles of the systems, as well as the reusability and scalability.

Manufacturers and designers hope for a system that allows for an incremental increase in both capacity and functionality of the system when required. Unlike DMS and FMS, the RMS offers such benefits [3].

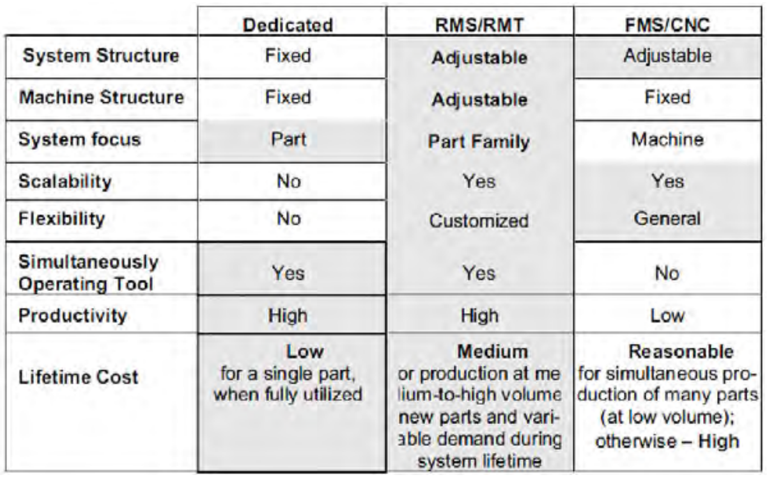
"*Highly productive and cost effective systems are created by 1) part family focus and 2) customized flexibility that enables the operation of simultaneous tools. The flexibility of RMS is customized flexibility, and provides all the flexibility needed to process the part family, and therefore is less expensive than the general flexibility of an FMS."* [15]

From these definitions and explanations, it is evident that there are some similarities between IMS, FMS and RMS, and it may even be argued that the RMS is merely a continuation of a FMS. However, there are key differences between RMS and FMS. An FMS system has the ability to manufacture small quantities of a variety of products [3]. Whereas from the start, a RMS is designed to be flexible in all aspects of hardware and software thereby allowing manufacturing of new part families which require totally different processing requirements in addition to production of a family of parts, while maintaining a high level of throughput at all times [4]. IMS on the other hand aims at constantly being at the optimum machine structure throughout its learning abilities, performance monitoring, reconfigurability and modular structure.

In order to support the aforementioned explanation, a comparison between the key characteristics of DMS, FMS and RMS is illustrated in Table 2. As demonstrated in the table, for a system to have a high level of reconfigurability it should have an adjustable structure, be scalable and flexible.

The number of research and development attempts for modular machines in recent years further strengthens ElMaraghys emphasis that modular machines are essential enablers for RMS. The following research and development of modular reconfigurable machine systems has been conducted over the past few years [12]: Flexible Manufacturing System Complex by MITI in Japan, the European Modular Synthesis of Advanced Machine Tools project by Hanover University, The Special Research Program 467 based at the Stuttgart University, The Engineering Research Centre of Reconfigurable Machining Systems at the University of Michigan and others.

Table 2: Comparison Between DMS, RMS and FMS [9]



	Dedicated	RMS/RMT	FMS/CNC
System Structure	Fixed	Adjustable	Adjustable
Machine Structure	Fixed	Adjustable	Fixed
System focus	Part	Part Family	Machine
Scalability	No	Yes	Yes
Flexibility	No	Customized	General
Simultaneously Operating Tool	Yes	Yes	No
Productivity	High	High	Low
Lifetime Cost	**Low** for a single part, when fully utilized	**Medium** or production at me lium-to-high volume new parts and vari- able demand during system lifetime	**Reasonable** for simultaneous pro- duction of many parts (at low volume); otherwise – High

Jimenez and Salinas [8] highlight that RMS are crucial for continuous improvement of manufacturing in the modern day global markets to match product changes and demand. However the unexpected evolution of the market means that a competitive advantage of RMS systems is not necessarily guaranteed if RMS systems are considered standalone systems.

Jimenez and Salinas note that although RMS indeed improves manufacturing system lifecycles, they need to be linked and developed on the current manufacturing practices. As the linking of these new RMS to the manufacturing plant as a whole is critical to the performance of RMS [8]. Therefore RMS are not stand alone systems or an end in itself, but for high performance and extended system life cycles, there needs to be a strong link between the current manufacturing plant systems and practices, and the continuously improving RMS system. Based on this, it can therefore even be argued that RMS is but a stepping stone towards the realisation of IMS.

Maier and Schroeder [16] have highlighted that the missing link between manufacturing systems in the plant and the supporting and linking plant technology can contribute to the failure of new manufacturing systems.

It is not practical, feasible, or financially viable for manufacturers to simply replace all of their manufacturing systems with RMS systems. As a result research has been directed at developing RMS systems that can utilise Commercially off the Shelf (COTS) modules for RMT [17]. Similarly, the current manufacturing systems in a plant and the linking between systems, needs to be considered in detail before RMS systems can be installed and implemented.

Furthermore an additional essential enabler for a RMS, much like IMS, is the software sub-systems or OACS for a RMT as highlighted by ElMaraghy [4]. Although attempts to develop standards for OAC such as: the Open system Environment for Manufacturing and the Open-controller Technical Committee in Japan, the Open Modular Architecture controllers in the USA, the Open System Architecture for Controls within Automation Systems in Europe [18], the Next Generation Controller project, the Enhanced Machine Controller [19] and others have been made, no unified standard has been accepted and sufficiently developed.

In summary, in an environment where the demands are so dynamic, a manufacturing system needs to be able to cope with these market requirements to prolong the beneficial operation period of the machine. This has led to the research of and motivation for the development of a Reconfigurable Manufacturing paradigm. However the limiting factor to the success of RMS currently is the lack of sufficient development of RMT and OACS which has been the primary motivation for this research

2.5 Manufacturing System Life Cycles

Due to the rapidly evolving market and consumer demands, product life cycles and, new technologies, manufacturing systems have shortened life cycles. Coupled with the global market and increased competition across the manufacturing industry, the ever evolving needs of the manufacturing industry means that the new manufacturing technology needs to be designed in such a way that the manufacturing system has a prolonged life cycle. The shortened product life cycles means it is too costly and inefficient to be constantly changing and redesigning the manufacturing systems for each new product in the market.

RMS are an example of systems which have the capacity for a prolonged life cycle. RMS accommodates for upgrades such that new functionality and modules can be added on. This flexibility allows for modification of the production capabilities, prolong the life span of the machines and reduce the capital investments required without having to introduce a totally new machine [4]. Figure 3 [14] illustrates how the RMS can be reconfigured during its lifetime to adapt to the production needs in terms of throughout and variations of manufactured goods. This reconfiguration offers important economic benefits to companies using RMS. Furthermore RMS aims is to reduce both the down time for reconfiguration and the ramp up time to production thereby furthering economic benefits.

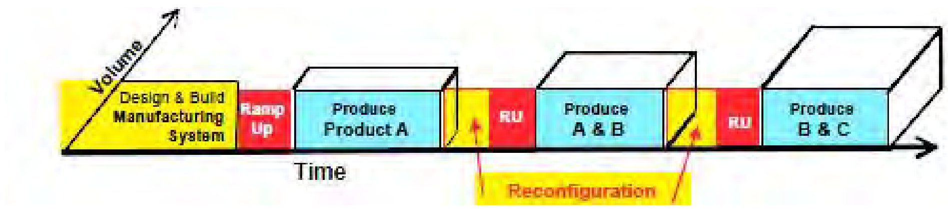


Figure 3: Illustration of the Reconfiguration of a RMS

2.6 Chapter Summary

Chapter 2 has presented a literature review analysing existing manufacturing paradigms, namely the DMS, FMS and IMS. Due to the inability of these systems to meet the current manufacturing demands, the RMS paradigm has emerged. RMS is expected combine the benefits of DMS, FMS and IMS to achieve a system that can constantly adapt its functionality through a reconfiguration process of both hardware and software. It is noted that although RMS has been researched, the two factors which are limiting the realisation of RMS is RMT and OACS which has led to the motivation for this research.

3. Open Architecture, Modular and Distributed Control Systems

The review of manufacturing systems in Chapter 2 has revealed that the lack of research and development of OA systems is currently one of the limiting factors for the realisation of RMS. To develop a framework for an OA system, a review of the aims and standards for OA systems is presented. This is followed by an analysis of industrially available Distributed Control Systems (DCS) and industrially available modular controllers in a bid to understand what the basis of design is for industrially available solutions.

3.1. Essential Characteristics of Open Architecture Controllers

The flexible nature of RMS requires equally flexible control systems that can quickly and reliably adjust its control functionality depending on the systems available hardware modules at any given time. Mehrabi et al. [3] have noted that software issues proved to be the area of greatest concern for the successful development and implementation of RMS.

For the realization of RMS, the system is required to be open at all three levels, namely: system, machine, and control. According to Koren [19] , the introduction of new modules in to a system may :

- Enable the production of new products on an existing system
- Improve the quality of production on the current system;
- Decrease diagnostic times;
- Lower the integration cost of discrete logic.

Furthermore, the addition of new modules may also:

- Lower the capital investments for system upgrades when new parts are required to be produced.

The technical committee of open systems from Institute of Electrical and Electronic Engineers (IEEE) define an open system as a system that: "*provides capabilities that enable properly implemented applications to run on a wide variety of platforms from multiple vendors interoperate with other system applications and present a consistent style of interaction with the user*. " [18]

The openness of the system is key to the effectiveness of the overall system, where openness is characterised by [18]:

- Portability:
  - The re-use of modules on different platforms without the need for modifications while maintaining the original functionality.
- Extendibility:
  - The ability of a number of modules to run on a system without conflict.
- Interoperability:
  - The integration of modules such that the modules function in a consistent way and communicate via pre-defined protocols of data exchange.
- Scalability:

- The ability to add, integrate, remove or swap in new modules thereby scaling the system up or down to adapt the performance or functionality of the system.

Further to these characteristics of an open system, there needs to be a strong link between the software and hardware sub architectures of a RMS [11]. Similar to the modular hardware architecture of a MRMT, modularity is also a key characteristic for the openness of the software system [19] where modularity is defined as follows:

- The software parts of the system are created in modules with well-defined interfaces such that a number of these modules may be integrated to produce a desired functionality.

Figure 4 [10] illustrates how these criteria affects the system both internally and externally.

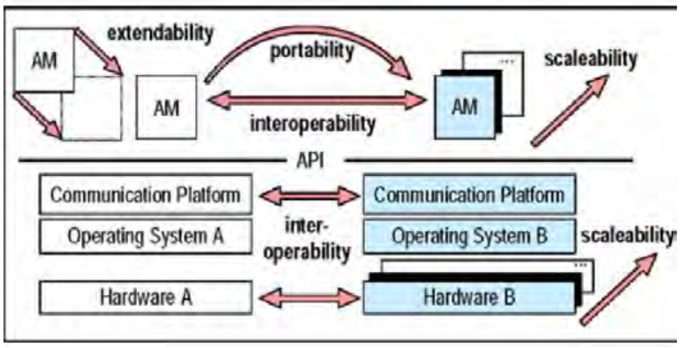


Figure 4: Criteria for Open Control Systems

Bearing in mind one of the aims of OACS is to allow a user to be able to integrate user specific algorithms and programs, the user will require access to the internal data structures and variables in order to implement such control algorithms. Koren [19] highlights two basic types of controllers:

- Vendor Specific (VS) controller:
  - A system which is designed by a specific vendor and the possibility for expansion only exists with the integration of modules and algorithms that are supplied by that vendor.
- Vendor Neutral (VN) controller:
  - An open system designed by various vendors with the aim of allowing future integration of modules and algorithms that can be supplied/developed by any vendor/end user.

On comparison of the two basic types of controllers:

- The closed architecture of a VS controller limits the possibility of future expansion unless the system uses VS products.
- Open controllers aim to remove these VS limitations and introduce standardization across all platforms
- Unlike the VS controller, the architecture of the VN controller is known; therefore any end user/control vendor may develop and integrate new methods and algorithms.
- End users working on a VN system will have the option of choosing new modules or algorithms from a number of control vendors allowing the best algorithms to be added to a system.

Pritschow et al. [18] and Koren [19] further note that for the success of OA systems, the system must be:

- Vendor neutral;
- Based on well-established open architecture standards;
- Provide well defined methods for data exchange and control reconfiguration.

Based on this it is evident that the development of a VN controller is a key enabling technology for an OACS as it enables users to apply specific control algorithms and programs.

The architecture of the control system should be designed to allow the user to add, swap or integrate new modules at any given time as illustrated in Figure 5 [19]. A VN system will support this as it will allow the integration or addition of third party modules.

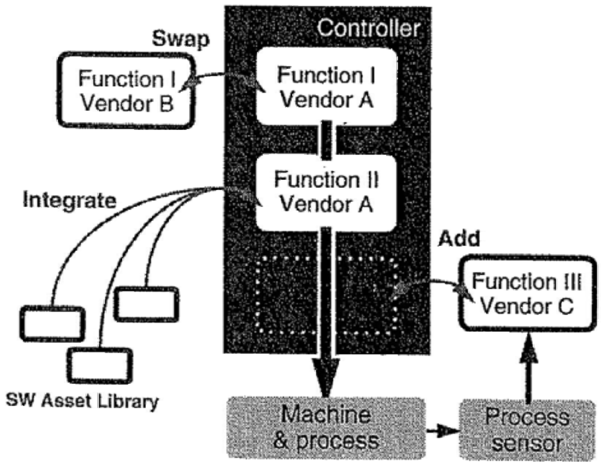


Figure 5: Three Critical Aspects of OA Controllers- Add, Swap and Integrate

Another key aspect for open systems is the reusability of basic modules in the creation of more complex algorithms via the integration into larger modules [19]. For example a basic limit switch module may be used in a position control module which in turn is used for an interpolation algorithm. To accommodate for the effective re-use of software modules, a well-defined API is required for each module. The API for each module defines the way in which the module interacts with other modules [19].

Pritschow [18] emphasizes that the performance of the control system is influenced by the level of interoperability between the basic modules. Since basic modules are used as building blocks for the system, the development of well-defined API between the modules is crucial.

Most of the OA systems that have been developed are based on PC hardware and standard operating systems [20] such as Windows or Linux. These have real time processing capabilities. These PC based controllers have the ability to support commercially available peripherals. For example, interaction with external electronics and control modules can be implemented via I/O cards which are readily available for PC's.

Since the robot programming languages are low level languages, the add on software development packages on PC's such as Microsoft Visual C#, Nokia's QT Designer and others can be utilized to ease the implementation of a PC based control system [20]. The OSACA Reference model in Figure 6 [21], presents an architecture design for the implementation of an OACS based on PC hardware.

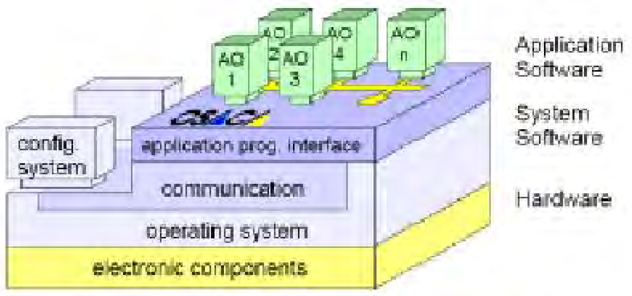


Figure 6: OSACA Reference Model

The system architecture describes how the control application is built on the standard PC hardware. The specifics of the computing system are encapsulated and is unnecessary knowledge for users. This encapsulation assists in control system portability and the interoperability between application modules [18].

Birla et al. have reviewed the Open Modular Architecture Controller (OMAC) API and concluded that the OMAC API seems the likely standard for the future [22]. The review covers the details of the API allowing third parties to: understand the standard and also to develop add-ons or to modify the current controllers using the OMAC API.

Further to the benefits of reconfigurability and scalability, the introduction of open control systems for RMT also assists in the cost reduction of systems upgrades, add to the flexibility of manufacturing systems in manufacturing new products and assist in prolonging the life cycle of RMS.

Since software issues proved to be the area of greatest concern for the successful implementation of RMS [3], this introduction to OA systems has emphasized key standards and features of OA systems and pointed out how the OACS will benefit RMS.

3.2. Emerson DeltaV

In a bid to understand what characterises industrially available DCS and modular controllers a review of two of these systems follow.

Emerson's latest DCS technology, DeltaV digital automation is a distributed control system designed primarily for the process industry. However the ideas of a distributed control system that is scalable, interoperable, has embedded intelligent control and the ability for third party integration is very relevant and applicable for an OACS design. The DeltaV system incorporates the following key features [23]:

- Input/Output(I/O) on demand:
  - Input and output signal adaptability with easy online integration
- Inherent scalability:
  - Well defined interfaces between all modules to allow scalability throughout the DCS system as a whole
- Embedded intelligent control:
  - Smart devices to ensure process performance and monitoring, as well as a choice on controller options for optimum plant performance
- Inherent integration for maximum interoperability:

- To allow third party standard based add-ons to the software

Figure 7(a) [23] illustrates the DeltaV S-Series architecture divided into three levels by defining interfaces between each level. The top level contains the application station, the middle level contains the distributed controllers and input/output cards and the lower level the sensors and equipment.

The operation principle behind the three levels and the distributed control is that the control programming and configuration occurs on the application station. When the configuration and programs are entered, they are downloaded to the distributed controllers.

Each distributed controller then interfaces its sensors and equipment, via the I/O cards. The controllers also control the equipment according to the programs downloaded onto the controllers. The controllers periodically send status updates and data to the application station for monitoring. Finally, at the lower level, the sensors are controlled by the distributed controllers and periodically send updated sensor information to the controllers.

Furthermore, the latest DeltaV S-Series hardware is based on CHARM technology, where a CHARM is a single channel card which can be for a digital or analog signal that connects the controller to the external hardware, depicted in Figure 7(b). Any program input or output can be mapped to any CHARM location which then links the control system to the field devices. This technology shows that even a large system can be modular and not pre-determined in its electronic construction. Additionally during operation, new I/O can be added to the system, and live time scans can be conducted to detect new signals. These scans can be done without affecting the existing system operations demonstrating a plug and play system which is scalable through a well-defined modular structure.

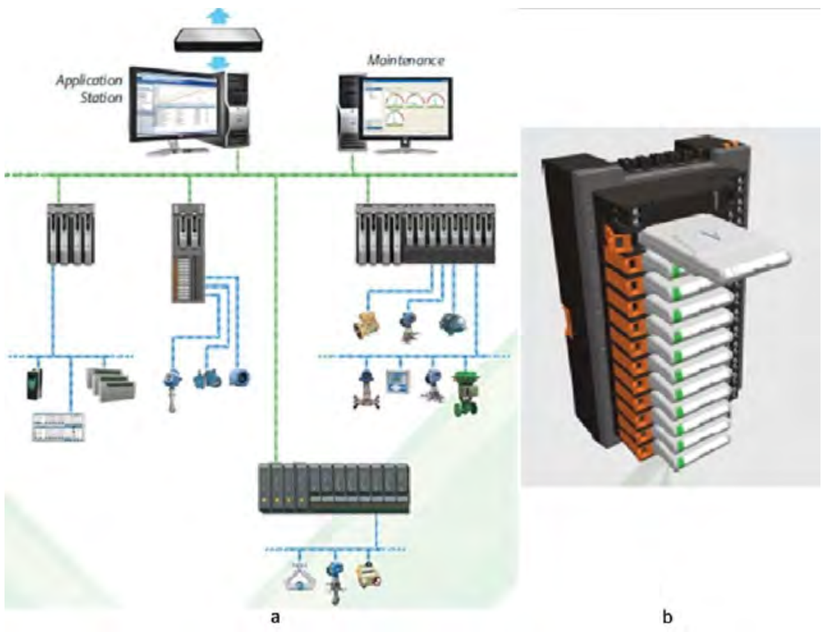


Figure 7: (a) DeltaV System Architecture (b) Plug and Play Modular Cards

Based on these points, although Emerson's DeltaV is designed for the process industry, the ideas of distributed control, scalability, interoperability and reconfiguration based on demand is applicable to RMT and has influenced the design of an OACS.

3.3. Siemens SIMATIC

Another industrially available product which has influenced the OACS design is the modular SIMATIC controller from Siemens. The SIMATIC controllers form part of Siemens Totally Integrated Automation (TIA), which aims to make a control system last the complete lifecycle of a machine or a plant by means of its modular structure. The SIMATIC controller can be based on a Programmable Logic Controller (PLC) based system or a PC based system [24].

The SIMATIC controllers are based on different hardware and software architectures giving the user a choice of different design modules to use. The controllers allow users to run custom programs on any of the devices as all devices: whether controller or peripheral, are mutually compatible. Furthermore, the SIMATIC controllers can be modified, upgraded or expanded using a number of compatible plug and play modules such as [24]:

- Modular controllers:
  - PC based
  - PLC based controller
- Input/output modules:
  - Digital/Analog Input cards
- Function modules:
  - Memory expansion module
  - PID control module
  - Position detection module
  - Motor Driver module
- Communication modules:
  - Profibus
  - Profinet

Similar to RMS, the SIMATIC range has been designed to allow the system to respond to competitive pressure and ensure operation over the lifecycle of the machine [24], although it has the limitation of being a VS solution.

To achieve this modular design throughout the controller architecture, Siemens implemented a well-defined data transfer interface that is transparent at all automation levels. In addition the SIMATIC system incorporates online diagnostic features to fault find and monitor the systems health. These diagnostic systems also contribute to easy plug and play configuration when the system is modified by minimising down times during system changes.

Furthermore, the SIMATIC system takes the modular approach not only in hardware design, but also in the software design and management. The SIMATIC system creates software blocks and allows users to duplicate and re-use code with ease. The reconfiguration and modification of the existing system can be done quickly and easily due to the well-defined modular nature of the systems hardware and software architectures [24].

Based on the above mentioned points, despite the Siemens SIMATIC controllers similar to DeltaV being VS solutions, they exhibits key characterises that are desirable for an OACS and has influenced the design of an OACS.

The research has also reviewed the following OA system development attempts: the PC based OA software system-CNC system [25] ,OA platform for the PUMA robot System [26] ,OA system for reconfigurable hardware-software multi-agent platform for CNC machines [11] , a reconfigurable and modular OAC: the new frontiers [26] , Simulation and implementation of and OAC [27] . Each of these have adopted different design approaches to achieve the similar objectives with varying degrees of success. Based on the review of these design attempts by academia, it is evident that no unified system for the development of and OACS exists. Although, research has also discovered that there are certain design ideas such as embedded distributed modules, a plug and play interface and a one to one software and hardware mapping are common to many of these OA systems and is central to the realization of a practical OACS.

On the contrary the tried and proven systems such as Emerson's Delta V charm based solutions and Siemens SIMATIC systems are proprietary and not open. Whereas other proprietary control systems are either not plug and play or do not allow for reconfiguration and adaptability over time as required by RMS [25].

In summary, the review has found that there are certain design ideas such as embedded distributed modules, a plug and play interface and a one to one software and hardware mapping are common to many of these OA systems and is central to the realization of a practical OACS. However for the true realization of an OACS, a standard need to be adopted which should have the necessary guidelines and support structures for OACS development to be practical and effective.

In conjunction with the above, these findings of the industrial solutions, DeltaV and SIMTIC modular controller as well as the findings from the review of other academia attempts at developing an OAC has led to formation of the basic ideas from where the OACS design has been derived.

3.5. Chapter Summary

Chapter 3 has covered a review of industrially available distributed and modular controllers followed by a literature review in which the standards and aims of OACS have been presented. A comparison of these has revealed certain key similarities between the various systems and these ideas will be central to any new OACS development. These findings and principles form the basis of the design of the OACS covered in Chapter 5.

4. Mechanical Systems

Chapter 4 presents the core ideas of CNC machining systems to assist in understanding of the functioning of the MRMT as similar design principles are adopted for the MRMT. Thereafter the design principles for a MRMT are discussed. This is followed by a review of the engineering and design of the library models that were used as the test rig for the OACS.

4.1 Computer Numerically Controlled Machines

CNC machines are mechatronic machines which produce products or parts automatically using minimal human intervention during operation. Numerical Control (NC), the heart of CNC systems incorporates a control system that activates servo motors through a driving mechanism to machine a part [28]. CNC machines aim to perform complex or simple machining tasks rapidly and precisely each time. In the 1970's the introduction, advancement and availability of microprocessors and memory occurred, leading to a better control system implementation for machines and this has allowed CNC machines to evolve to mechatronic systems. Prior to this, machines were hardwired and only had NC systems.

CNC machines comprise of a mechanical component and a numerical control system, which is the electrical component [28]. A CNC machine can perform a number of operations on a part or product depending on the type of CNC machine. Examples of different CNC machines include: drilling, milling, turning, and welding as well as painting, shearing, boring and grinding. Figure 8 [28] illustrates the architecture of a NC machine tool with the machining operation flow.

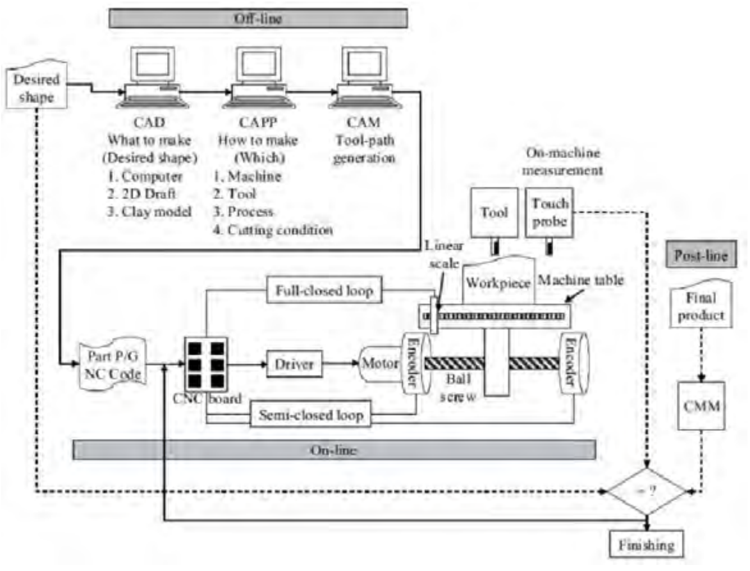


Figure 8: CNC System Architecture

From Figure 8 it is apparent that a CNC system consists of an offline and an online section. The offline section consists of: the CAD, Computer Process Planning (CAPP) and CAM. Firstly a part that needs to be manufactured will have a 2D or 3D CAD model. Thereafter the CNC system plans the machine structure and generates the necessary information for machining. Finally the CAM is run to generate the part program based on the CAD models and CAPP information.

The online part of the CNC system facilitates the actual machining of the part. The part program generated by the CAM is used as the input for the machining. The part program is first read and interpreted. Then, based on control theory, the machine axes are actuated via motor driving circuitry and servo motor to machine the part. Live time and online monitoring of the machining process is conducted, and where necessary, corrections are made by the CNC system to ensure accurate part machining.

The CNC system interprets the part program and converts this to commands for servo motor driver circuitry. The servo motors then converts these commands to electrical signals that power up the servo motors. The servo motors are coupled to power screw couplings, the rotation of the power screws translates to linear movement of a nut which in turn translates to movement of the work piece or axis as depicted in Figure 9 [28].

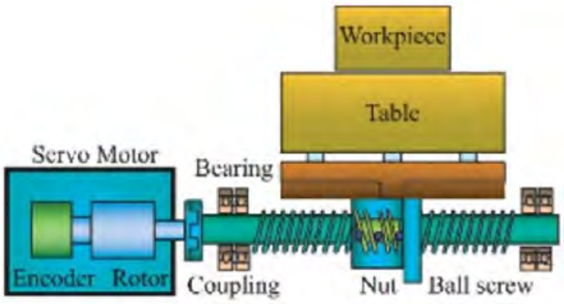


Figure 9: CNC Servo Driving Mechanism

As discussed in Chapter 3, the CAM and the online features of CNC machines have been researched, designed and implemented as they are relevant to this research. The CNC machine tool, like the machine axes, is based on the following: a servo motor capable of high speed, good acceleration and deceleration, wide speed range, quick response times and high torque. Depending on the type of machining, the machine tool will either be directly mounted on the CNC machine or coupled with a spindle driving mechanism based on a pulley and a belt.

4.2. Modular Reconfigurable Manufacturing Tool

The concept of MRMT has been derived from the basis of RMS which aims to combine the benefits of DMS and FMS. No set of agreed principles exists for an RMT and the option is to have a system with rigidity, such as DMT or to have a system with the flexibility of a CNC system, or even to have a system with flexibility that exceeds the flexibility of CNC systems.

Machines that are rigidly designed face the risk of becoming redundant if the market demands change significantly. In contrast, machines which exceed the flexibility of CNC machines are often too expensive, as discussed in Chapter 2, and therefore unfeasible and impractical for to design and implement.

RMS and RMT need to be inherently capable of adapting based on the current market demands and manufacturing requirements. Based on this, and to ensure that the RMT is neither too rigid nor too customized, the concept of the MRMT has been introduced.

The idea of a MRMT is to allow a machine tool to be implemented as required at the present time, and as the production, manufacturing and machining requirements change in time, the RMT can be modified by adding/swapping/removing modules on the system through a reconfiguration process. However, for the realization of MRMT, the MRM needs to be completely modular and reconfigurable in the mechanical and control architectures.

The concept of a modular machine has been researched, and there have been several attempts at developing modular machines. However, a fully functioning modular machine has yet to be been presented. This research was taken up as a MRMT had been developed in terms of mechanical architecture, but just as in other attempts, the lack of truly modular control architecture limited the functionality, operability and success of the MRMT.

The MRMT has a set of Library modules that allow rapid development and reconfiguration of the MRMT to be able to respond to production requirements. The MRMT library modules [6]:

- Exist at physical level;
- Provide a solution that allows the complete synthesis of a modular machine;
- Are assembled and disassembled in a building block manner;
- Contain all electro mechanical component;
- Enable simple and easy after-market reconfiguration.

Figure 10 [6] shows a conceptual library of MRMT modules. Library modules can be classified as motion, process or accessory modules. These can be defined as follows:

- Process modules are the tools of the MRMT that provide the MRMT with manufacturing capability such as drilling, boring or cutting;
- Motion modules provide movement to the MRMT. They are combined to form the different axes of the MRMT and as a result influence the Degrees Of Freedom (DOF) available.
- Accessory modules don’t directly influence the process or movement of the MRMT, but they provide valuable support and assist in operation and efficiency of the MRMT. Clamping modules, steady rests, flow rests and work table modules are examples of accessory modules.

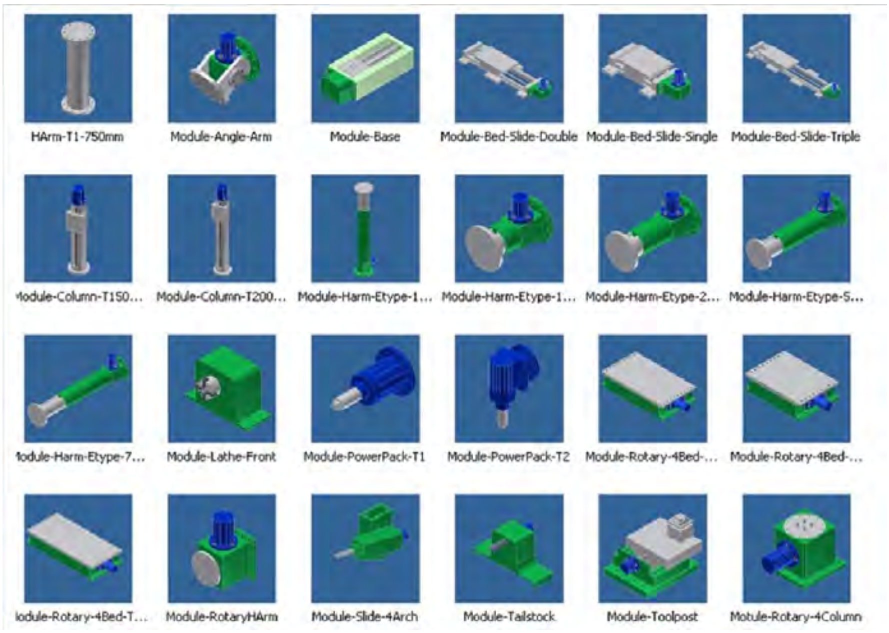


Figure 10: MRMT Conceptual Module Library

A combination of the library modules can be assembled together to form a kinematically viable MRMT. In terms of reconfigurability, by replacing process modules with another, the machining function of the MRMT will change. Similarly by adding, swapping or removing a motion module, the MRMT can be reconfigured to provide machine functioning in more DOF. Figure 11 [6] depicts how a 3-axis boring machine is reconfigured to a 4-axis milling machine by the addition of an additional motion module and replacement of the process module.

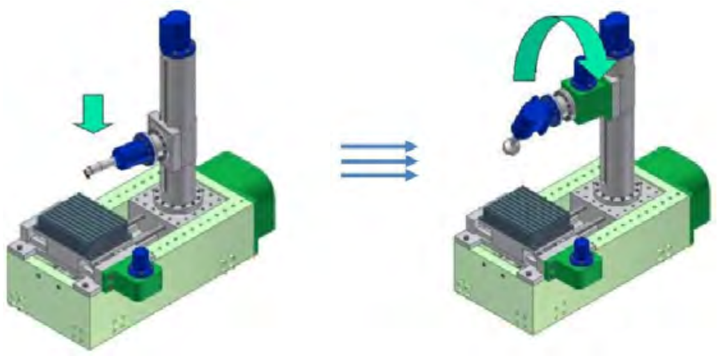


Figure 11: Machine Reconfiguration

The library modules of the MRMT were designed such that each module added a single DOF to the machine tool and each module contained all peripherals required for its operation, such as servo motors and sensors. A total of six modules, three linear axes (providing movement in the conventional Cartesian planes, X, Y and Z), and three rotary axes (providing rotations about these planes, A, B and C), from the conceptual library were designed and constructed for use with the MRMT.

Furthermore, the MRMT modules were designed with a consistent set of bolted interfaces such that modules were interchangeable to maximise machine reconfigurability. The bolted interfaces enable torque and force propagation throughout the machine structure and have the strength to support the shear and tensile forces during machine operations [6]. Figure 12 [6] illustrates the two bolted interfaces on a module; allowing different modules, with either of the interfaces to interface this module.

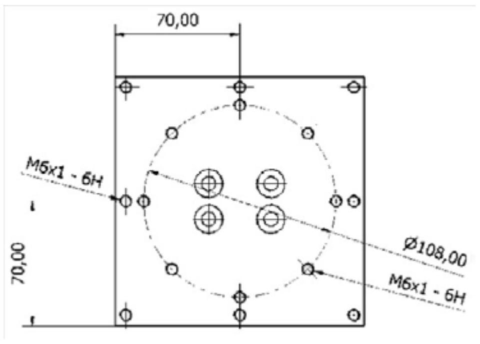


Figure 12: Drawing Indication: Two Bolted Interfaces on Mechanical Module

For operation of the MRMT, or any machine tool for that matter, a practical position reference methodology is required. The MRMT uses the conventional right hand Cartesian coordinate system which covers all modules, linear and rotary, as shown in Figure 13(a) [29]. This coordinate system needs to be fixed to an absolute position reference so that all modules can operate in respect of a global location. The absolute position reference is located on the work table where the machining would take place and is show in Figure 13(b) [29]. A further discussion of the representation of the position of the axes and reference positions is presented in Chapter 7 along with the kinematic computations for the MRMT.

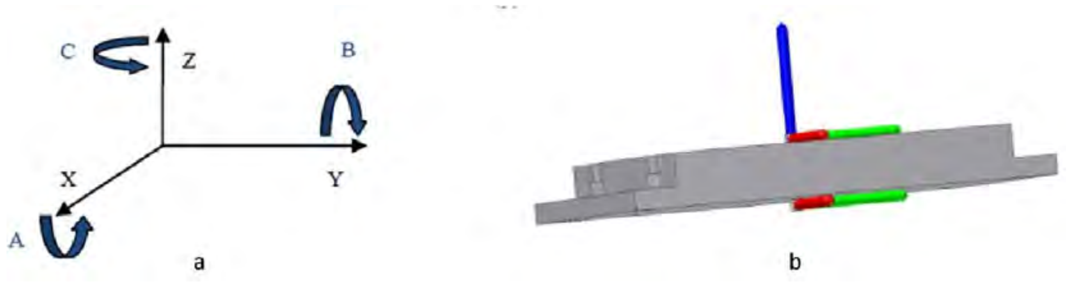


Figure 13 (a) Coordinate System (b) Absolute Position Reference

The MRMT was designed and constructed with six DOF and two machining functions, namely drilling and turning. The purpose of this research, as discussed in Chapter 1, is to prove the OACS concept on the existing MRMT. Therefore all the axes and machining functions were not utilised. Only three axes, Z, Y and C, and the drilling tool module were used to implement and test the OACS. It must be noted that the axes were renamed from Y, Z and C to X, Z and A, due to the simplified implementation with the aim of proving the OACS concept. Following from this, A represents the rotation about the Z Axis.

If need be, the OA nature of the control system will allow easy reconfiguration to upscale the machine structure to incorporate additional axes and machining functionality. The MRMT modules used will be discussed further as this will assist the reader in gaining an understanding of the MRMT platform which the OACS has been designed on. As mentioned, the axes chosen were two linear axes, X and Z as well as one rotary axis, A. The linear axes are controlled in the same manner as the linear axes on a CNC system as discussed Chapter 4.1.

The X axis forms the base module of the MRMT, and therefore needs to be strong to support the remaining models, and to dampen the vibrations from movement. The X axis module also has the platform for the work piece to be clamped on. Figure 14 [29] is an illustration of the drive mechanism. The following list summarises the design of the X axis module [29]:

- Manufactured from steel to provide the necessary strength;
- Drive mechanism is based on a steel slide driven by an ISO metric M24x3 power screw;
- Power screw is supported by two deep groove ball bearings at either end;
- Power screw runs in a brass nut housed in the slide;
- Two additional 20mm silver steel rods add support and rigidity for the sliding mechanism.

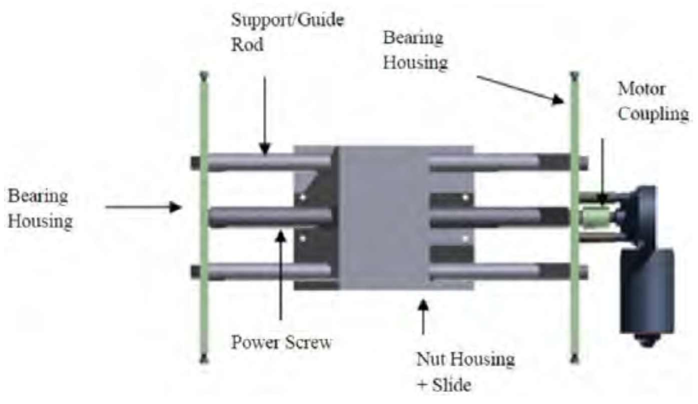


Figure 14: X Axis Drive Mechanism

The Z axis forms the column module of the MRMT, and supports the machining module, which is the drill tool piece for the MRMT. The drive mechanism is illustrated in Figure 15 [29]. The following list summarises the design of the drive mechanism [29]:

- Manufactured from aluminium (295-T4) to limit weight while maintaining strength;
- Drive mechanism is centrally located and based on a steel slide driven by an ISO Metric M24x3 power screw;
- Four silver steel rods support the guide mechanism for the slide;
- Power screw runs in a brass nut housed in the slide;
- Power screw is supported by thrust bearings at the ends of the column.

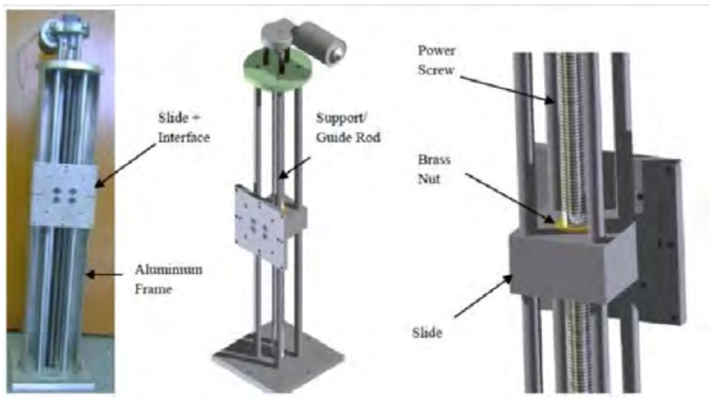


Figure 15: Z Axis Drive Mechanism

The C axis, a rotational axis which rotates about the Z axis, is designed to have the work table mounted on top of it. The drive mechanism is illustrated in Figure 16 [29]. The following list summarises the design conditions of the module [29]:

- Manufactured from aluminium (295-T4) to limit weight while maintaining strength;
- Drive mechanism has an upper section rotated by a reduction worm gearbox;
- Upper section is supported by thrust bearing at its base;
- Worm gearbox is coupled to the motor by a 12 mm steel shaft;
- Worm gearbox ensures that rotation only occurs by the motor and not due to load forces.

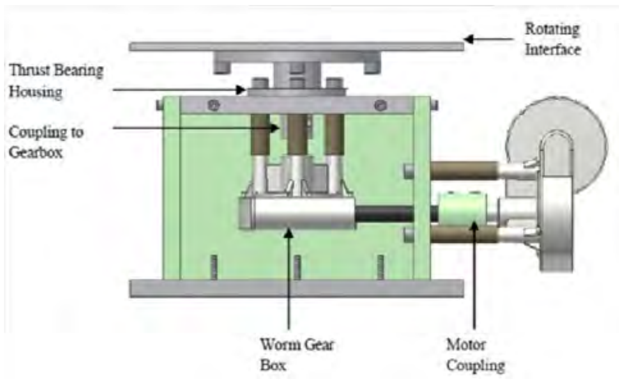


Figure 16: A Axis Drive Mechanism

A single process module, namely a drilling head, is used by the OACS test platform. The drilling head module incorporates a 12 V, 80 W DC motor and a spindle attached to an aluminium mounting. Figure 17 [29] illustrates the internal mechanisms of the drilling module. The following list summarises the specifications of the module [29]:

- The motor is coupled with a planetary gearbox;
- Maximum spindle speed of 580 rev/min;
- Internal gearbox has a back-torque limiter that connects the gearbox to the spindle;
- A slip clutch, being a back-torque limiter, prevents the motor stalling.

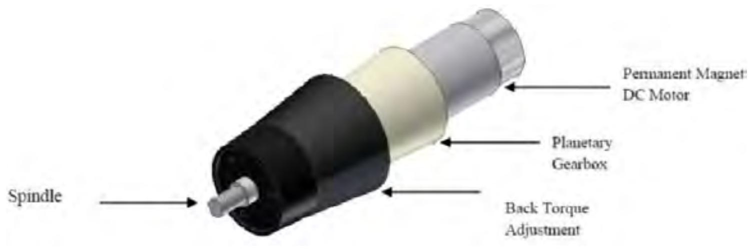


Figure 17: Drill Head Module

The Drilling head was attached to the machine tool via an accessory module, namely a modular range extension arm. The arm, a spool piece, was created to allow the MRMT process module to reach the workspace due to the length of the base module, as shown in Figure 18 [29]. The extension arm increases the movement of the drilling head by 350 mm.

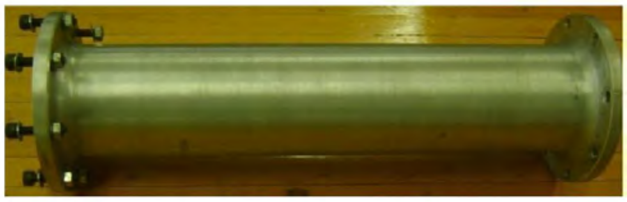


Figure 18: Accessory Module - Modular Range Extension Arm

Each of these modules, when assembled together, will need to be checked if they can work together. To do this check, each module has an associated transformation matrix which is used to check the kinematic viability of the machine configuration when the modules are assembled. The Kinematic modeling theory required for this is covered in Chapter 7 and in further detailed in Appendix A.

The MRMT was designed and developed as part of a previous research by Padayachee et al. [6]. The MRMT is designed from the base up to be reconfigurable, flexible, transformable, and scalable in its mechanical architectures. The development of a modular OACS for the MRMT was not part of the MRMT scope, and therefore the MRMT was designed and built with a fixed electronic and control software solution. When a specific software configuration was selected in software, it assumed that the mechanical modules were connected, and the associated electronics were ready and active. The set up was predetermined in software, and therefore, to fully realize the potential of the MRMT, and OACS which was modular and reconfigurable was needed. This then led to the motivation of this research.

4.4. Chapter Summary

A summary of the operation of CNC systems has been presented to assist in understanding of the functioning of the MRMT as similar design principles are adopted for the MRMT. Thereafter the design principles for a MRMT are discussed. This is followed by a review of the engineering and design of the library models that were used to test the OACS on.

5. Open Architecture System for a Modular Reconfigurable Machine Tool

The work presented covers the research aims and outlines that builds on the review of modular and distributed control systems to present a high level system design for the OACS. A core component of the research and design is the distributed modules and the discussion details how they add value to the system. This is followed by the introduction of buffer layers and how these layers add significant value to the reconfigurability and customizability of the system as a whole.

5.1 Research and Design Outlines

Mehrabi et al. [3] highlight that software issues represent the area of greatest concern for the successful development and implementation of RMS's. For the realization of RMS's, key technologies such as modular machines and OACS need to be developed. These OACS need to be designed with similar fundamentals to RMS.

The research development of this control system for the MRMT aimed at incorporating as many features of OA systems as possible. In the early stages of research, comparisons between RMS systems and OACS indicated a high level of correlation in the critical aims and features of both systems. The following key features of RMS systems were identified for development in the OACS:

- Software Modularity
- Reconfigurability:
  - Extendibility;
  - Interoperability;
  - Scalability.

These features would enable the control system to be open and allow for easy setup, configuration, expansion, and diagnostics as well as a decrease in configuration times. On a higher level, the following additional features were also incorporated into the design:

- User Customization:
  - User specific algorithms;
  - User specific programs.

These methods allow user specific customization, which is a key evaluation for open control systems. Users are able to create customized programs via a programming interface. The user specific programs and algorithms can be created and saved to be used again if necessary. The aim of user specific customization is to allow users to create user specific programs and algorithms depending on their MRMT structure, and consequently their control and manufacturing requirements.

5.2 Mechatronic Design Approach

For the benefits of a RMS system to be seen there needs to be a high level of correlation between software and hardware components of the machine. Figure 19 [30] represents the mechatronic design approach adopted for the development of the OACS.

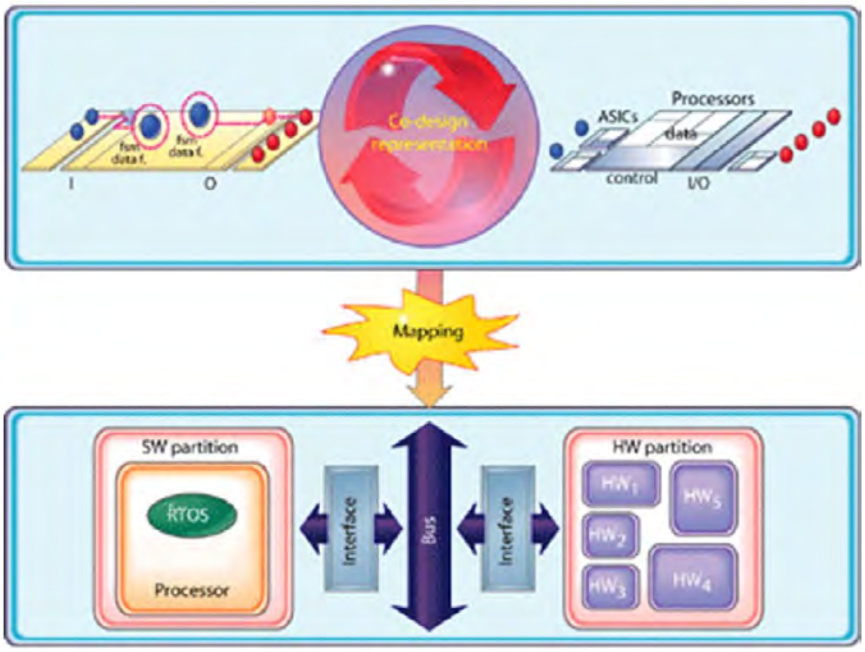


Figure 19: Hardware/Software Co-Design Approach

The mechatronic design approach separates the system in three individual components, the mechanical system, the electronics, sensors and actuators and information technology which is the software design and automation [31]. The MRMT and OACS combination form a mechatronic system with mechanical hardware, actuators, sensors, controllers and analog and digital electronics. The design of the OACS followed a mechanical hardware and software-electronics hardware co-design approach to ensure that the architecture of these systems correlated. The integration of the mechanical and electronic system must be considered as one system from the very beginning to ensure an optimum solution.

The system architecture, electronics and software were co-designed with the mechanical system at a high level to ensure a high level of correlation. This comprised gaining an understanding of what was required to achieve a one to one mapping in software and hardware such that the systems could be aligned. Thereafter, the design focused on mapping the high level architecture to physical implementation which comprised of the design of embedded hardware, a dynamic communications interface and modular software implementation.

5.2 OACS High Level Design

The research proposes a novel solution, attaching a microcontroller based distributed module to each of the mechanical modules on the MRMT to assist with reconfigurability and self-diagnosis as shown in Figure 20. In addition, the design focuses on modularity and through the object orientated C++ implementation of the host PC, the system creates a specific software module for each corresponding hardware module using classes in C++.

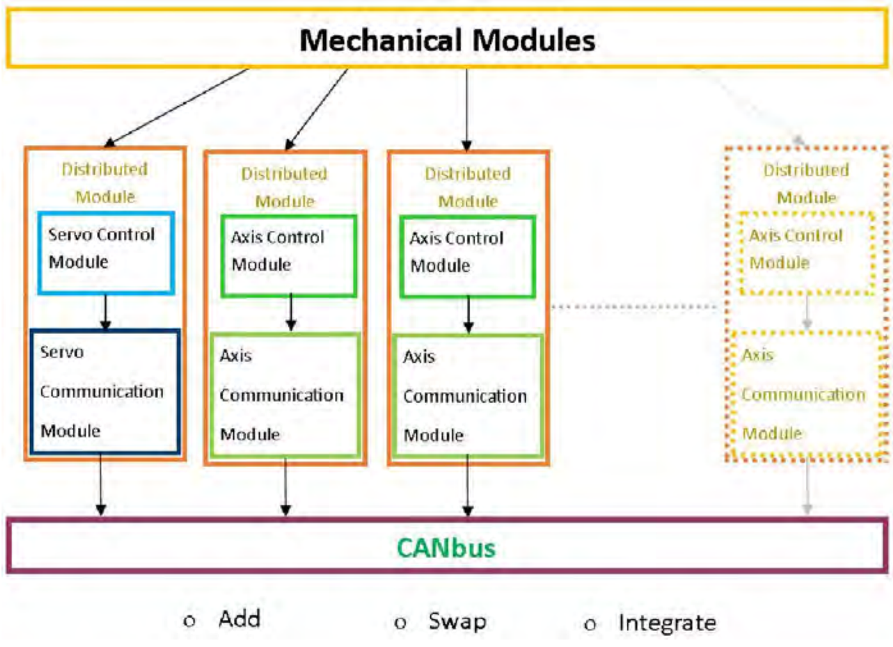


Figure 20: Integration of Distributed Modules

This one-to-one linking between mechanical modules, distributed modules and software ensured that the system was modular in all aspects, reconfigurable, extendable, and scalable. It also allowed for an easier set up after system reconfiguration. In addition, the distributed modules allow axes to be added, swapped or integrated with ease.

To ensure consistency in this modular approach throughout the design process, a co-design process was required, where the software on the host PC would need to be closely linked to the distributed modules. A high level system design for the OACS is shown in Figure 21.

At the head of the system is the MRMT and as discussed each mechanical module of the machine tool is connected to a specific distributed module which acts as an independent node on the CAN bus communications network. The bus network is linked to all the distributed modules as well as the host PC. The distributed modules are the interface between the host PC and the mechanical module on the MRMT. Each distributed module contains a dedicated microcontroller, and all associated electronic circuitry to control and communicate with the host PC.

The host PC is in constant communication with the distributed modules and initially data downloads are initiated from the host PC. The host PC downloads critical configuration and control limitations from the distributed modules. During operation the host PC transmits the control requirements to the distributed modules. Data such as distances to move and speed of movement are transmitted. Furthermore, tuning parameters, diagnostic and fault finding and performance evaluation is transferred over the bus. Therefore, a robust, reliable, interoperable and scalable communication system was required to handle the communication between the nodes on the MRMT.

The host PC processes the machining requirement for a part program, thereafter the control algorithms process the machining requirements, where the data is passed through interpolation, PID, acceleration and deceleration control routines. The memory of the host PC stores the control system, control algorithms, and software modules for each hardware module.

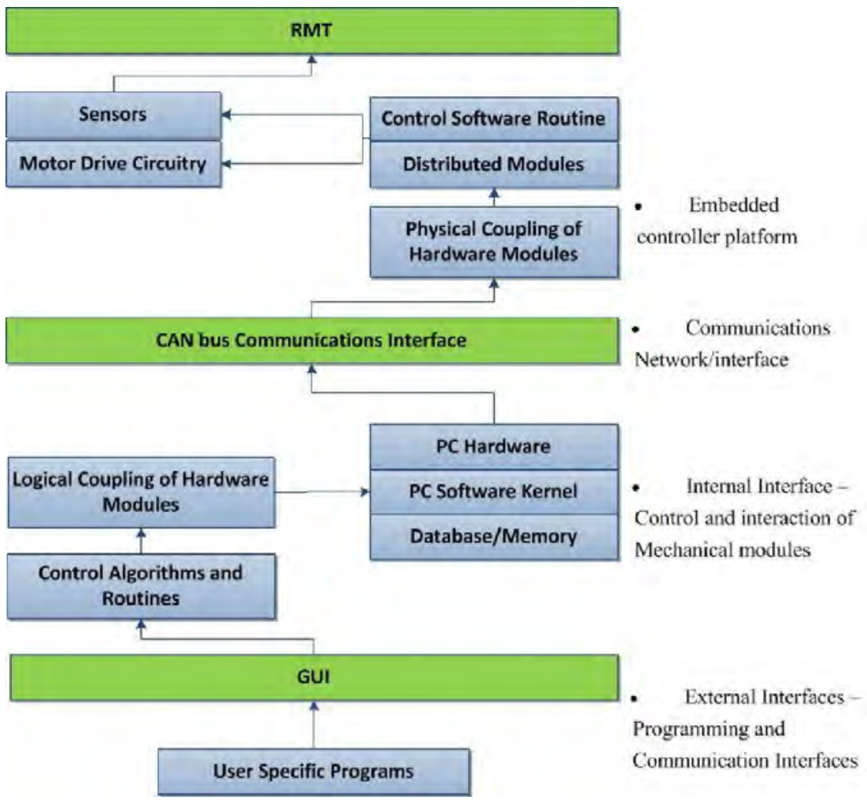

RMT
Sensors
Motor Drive Circuitry
Control Software Routine
Distributed Modules
Physical Coupling of Hardware Modules
• Embedded controller platform
CAN bus Communications Interface
• Communications Network/interface
PC Hardware
PC Software Kernel
Database/Memory
Logical Coupling of Hardware Modules
Control Algorithms and Routines
• Internal Interface – Control and interaction of Mechanical modules
GUI
User Specific Programs
• External Interfaces – Programming and Communication Interfaces

Figure 21: High Level System Design

The mechanical hardware modules of the MRMT are easy to visualize. The design aims to create a similar, non-visible, modular implementation in the software libraries of the OACS. The aim was to develop similar modules or containers of code for each mechanical axis and associated distributed module connected to the system, ensuring consistency in creating the one to one links for all sub sections as discussed.

In order to achieve modularity in software, the C++ programming language was used in the Visual C# programming environment. The object orientated approach of C++ allows the modules to be defined from generic classes with well-defined API.

The object orientated approach also allows for data abstraction and inheritance, which has been useful to control what is visible to the end user [25]. Furthermore, the inheritance features of C++ allowed the design to initially create one class for all modules. Thereafter during set up and configuration, when modules are detected and active, new classes are derived from the generic class depending on the type of module that is connected. Figure 22 further illustrates this idea, and the key points of the software system can be summarised as follows:

- Each hardware module has an associated class object that contains all the information with respect to hardware and control functionality;
- For example, as in Figure 22, each translational axis which is connected via its distributed module has a basic software module or class in the database;
- The distributed modules, contain the simple control algorithms such as collision detection, speed control, position control etc;

- A combination of the basic modules and the built-in simple control algorithms is used during runtime of advanced user specific programs or algorithms such as interpolation, acceleration and deceleration.

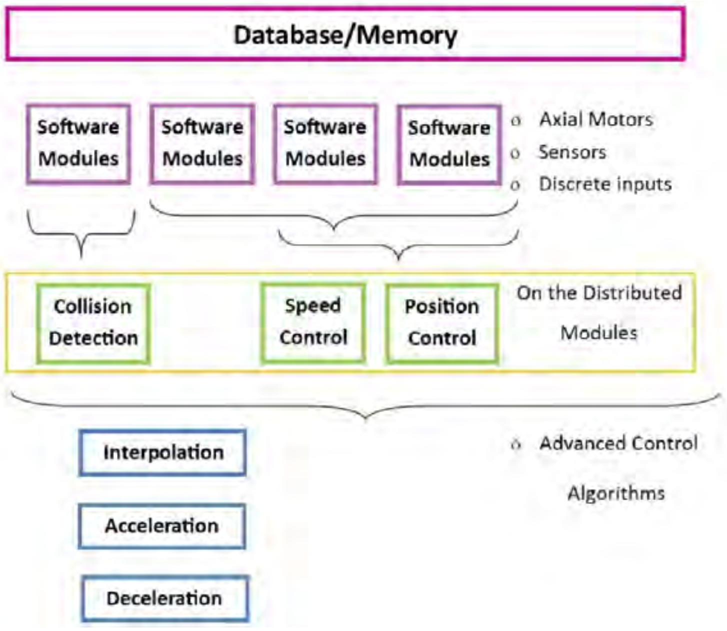


Figure 22: Examples of the Software Modules In System Memory

The final layer of the design is the GUI. The GUI is used to configure the MRMT, enter the physical configuration of the MRMT, specify and choose controllers and enter user specific programs. The GUI is also be used to display the relevant control processing and processed data to the user and allow the user to conduct debugging and diagnostics on the system.

5.3 Distributed Modules

As mentioned, the distributed modules are be the link between the mechanical modules and the host PC via a CAN bus interface. Each distributed control module contains all the hardware required for that specific module. Figure 23 shows an example of the components that are grouped as a distributed module, in this case, an axis/servo module.

The introduction of the distributed modules benefits the OACS in several ways:

- Since the distributed modules contains all the information specific to itself and the peripheral modules it interfaces, the host PC does not need to store all the information of the corresponding module, the host PC can rather download the information when required, thereby assisting in the dynamic reconfigurability of the system;
- The following list summarises the data that is be stored and downloaded from a distributed module:
  - Module ID;
  - Module type;
  - Maximum/minimum operation speeds;
  - Transformation matrix.

- Downloads can occur at start up, or during reconfiguration due to the addition or removal of a module;
- Since the distributed controller lies between the host PC and the corresponding axis or servo or sensor module, any type of module may be added to the system regardless of its communication protocols or control requirements;
- If any module does not comply with the standardized communication protocols or control requirements, a customized distributed module can bridge the gap and do the necessary conversion for the non-standard axis or servo or sensor module, making the actual module invisible to the host PC but ensuring system functionality;
- Similarly, a non-standard distributed module that may not have the correct communication module can be interfaced with an additional transceiver component that matches the standard communication protocol;
- These two points mean that the system has a level of invisibility between the layers such that as long as the basic standard is adhered to, any axis or servo or sensor or microcontroller can be used with interfacing circuitry to match the standard;
- In the case of an module being added or reconfigured, the distributed module microcontroller does not need to be changed, but rather allow for software to be reprogrammed or updated, decreasing the cost of upgrades or modifications to the system;
- Similarly, If the current distributed module microcontroller fails to satisfy the system requirements, the microcontroller can be replaced with a new one that can be customized to interface the existing axis or servo or senor, decreasing the cost of upgrades or modifications to the system.

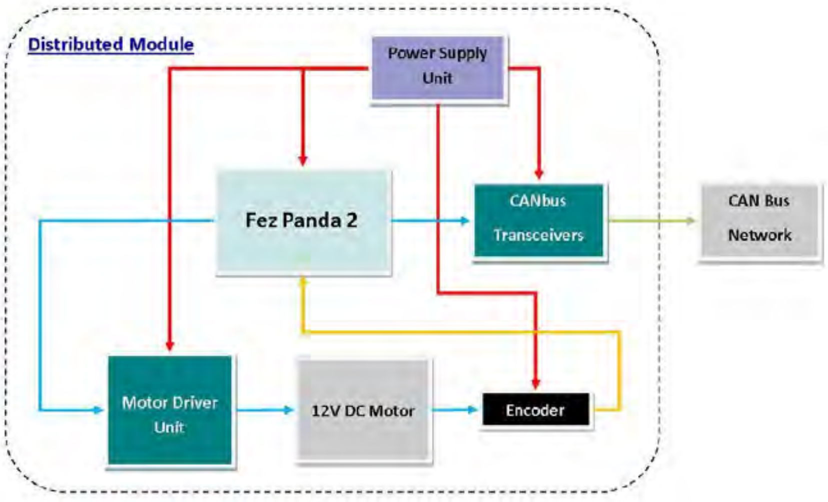


Figure 23: Example of the Components of a Distributed Controller

In summary, the introduction of distributed microcontroller based modules between the host PC and the corresponding actuator or sensor module has assisted the system by:

- Increasing flexibility;
- Reducing the time for reconfiguration;
- Assisting in the rapid introduction of new technologies/module;
- Reducing the cost of system upgrades.

Based on the aforementioned points, it can be seen that the introduction of distributed microcontroller based modules assists the system achieving the desired goals of an RMS and OACS.

5.4. *Buffer Layers*

The introduction of distributed modules in the OACS indirectly creates two buffer layers in the system architecture. These two buffer layers, between the mechanical modules and the distributed modules, and between the distributed modules and the host PC are depicted in Figure 24.

These two layers create a buffer layer or an invisible layer between each of the three tiers. This buffer layer allows for the components at each tier to be of many variations, types and architectures. Consequently all that is required to ensure system functionality is that the interface and data transfer between the tiers is consistent and standardised.

The functioning of the OACS is therefore independent of the types of modules at each tier or what is located at each tier. The components of a module in a tier can therefore change and be upgraded without affecting system functionality.

For example, on tier 3, the mechanical modules, a user may choose to add in induction motors instead of the 12 V DC servo motors. In this case, to ensure system functionality the motor driver and the associated I/O from the microcontroller need to be customized to interface the new motor.

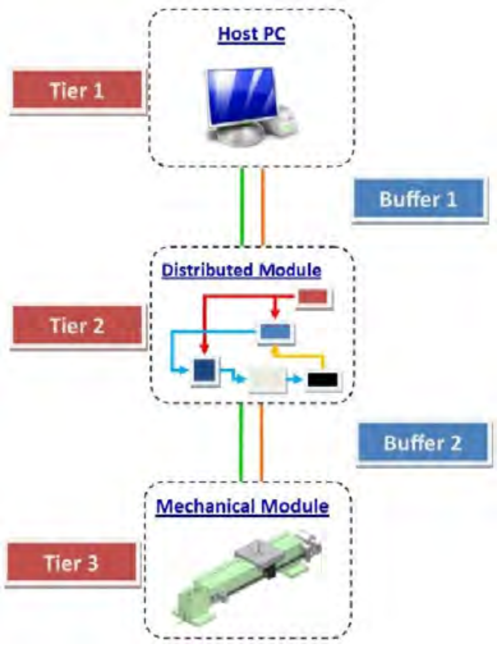


Figure 24: Module Linking With Buffers

Similarly, in the middle tier, the distributed module has an embedded microcontroller, this microcontroller can be of any type, make or from many manufacturer. The supporting peripheral electronic circuitry interfacing the microcontroller can also differ from other distributed modules. As long as the interface and data transfer protocols between each layer are adhered to, the components at each level can be customized to the user’s requirements.

To demonstrate this flexibility, the OACS has been designed and developed using three different microcontroller development boards, and with customization of the supporting electronic circuitry, the system has maintained complete overall functionality.

This design approach opens up the possibility for the integration of off-the shelf components into any MRMT. With customization of the components in that tier to ensure that the interface and data transfer between the other layers is consistent and standardised, the system maintained the required functionality as a whole.

5.5. Chapter Summary

Chapter 5 covers the high level system design of the OACS that has been tested on the existing MRMT. The work presented covers the research aims and outlines and builds on the review of modular and distributed control systems to present a high level system design for the OACS. A core component of the research and design is the distributed modules and the discussion detailed how they add value to the system. Finally, from the high level system design, the system is segregated between host PC, distributed modules and the remaining electro-mechanical sub-systems. The links between these layers act as buffer layers and add significant value to the reconfigurability and customizability of the system as a whole.

6. Electronic Subsystems

Chapter 6 covers the core aspects of the electronic sub systems of the research and design. The multiple microcontroller implementations of the distributed modules is motivated for and presented followed by a discussion presenting the communication interface, the sensors and the design methods to overcome false triggering and noise interference. Finally a high level overview of the power distribution network is presented indicating the key power distribution channels across the whole electronic subsystem.

6.1 Overview

The electronic subsystems of the research and design cover the various components that lie in-between the host PC and the mechanical modules on the MRMT. Figure 25 shows a high level diagram of the system architecture focusing on the key components that will be covered in the remaining sections.

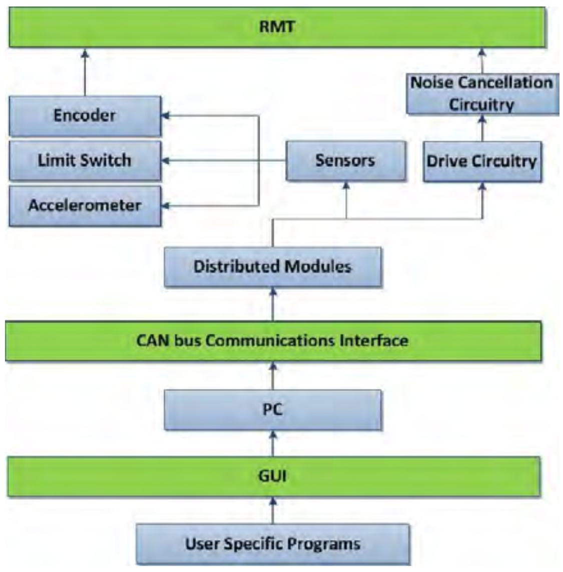


Figure 25: OACS Architecture

6.2 Host PC

At the head of the OACS is the host PC. The introduction of faster processers for computers and a reduction of prices and increased availability have allowed for an increased use of computers as PC-based controllers for CNC machines [32]. PC based controllers are generally flexible, open and can be easily integrated into multiple manufacturing configurations [32]. The open and flexible nature of PC make PCs ideal for use as a host PC for the OACS, as openness and flexibility are the central and core characteristics for and OA systems.

Users may use Microsoft's Visual C# express to configure, modify and program the external distributed control modules. In addition, the existing USB hardware further assists the end user with programming the distributed modules with a readymade plug and play interface. Furthermore users may use C# to edit, add in and then re-compile the application with user specific control algorithms, which is one of the fundamental aims of an OA platform.

The PC has been fitted with a CAN bus PCI card which eases the integration of the distributed modules as the 9 pole SUB-D connecter, which offers a plug and play interface with all external boards. The host PC communicates with all the distributed modules via the two wire CAN bus communications network according to the custom data packets, which are covered in section 6.4.

6.3 Distributed Modules

The distributed modules located at the different stations of the MRMT are designed around the FEZ Panda 2 board. After reviewing and analysis, the FEZ Panda 2 board was chosen for two primary reasons, firstly ensuring that the board had all the necessary functions required such as CAN bus, timers and interrupt based operations. Secondly, the FEZ Panda 2 board runs on Microsoft's .NET Micro Framework [33]. Since the PC application is designed using Microsoft's Visual C# which is also based on Microsoft's .NET framework, in order to have cross platform uniformity and standardisation, as well as ease of understanding the use of a common language and framework was preferable.

The FEZ Panda 2 shown in Figure 26 [33], is a high performance 72 MHz 32-bit USBizi ARM7 processor, with the following key features [33]:

- USB device connection for run-time debugging;
- 54x Digital I/O ports;
- 6x 10-bit analog Inputs;
- 10-bit analog output 6x Hardware PWM channels;
- 2x CAN channels;
- 4x UART serial ports;
- Built-in Real Time Clock;
- Multi-Threading.

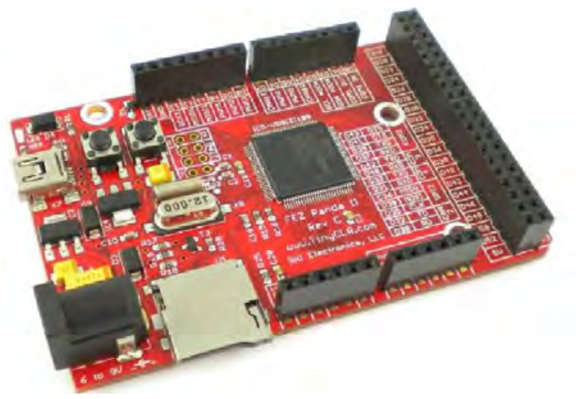


Figure 26: FEZ Panda 2 Board

The distributed modules are responsible for a number of tasks depending on its application. A complete schematic showing all signal and electrical connections is attached in Appendix B. Figure 27 and Figure 28 summarise the tasks of the spindle and servo modules respectively.

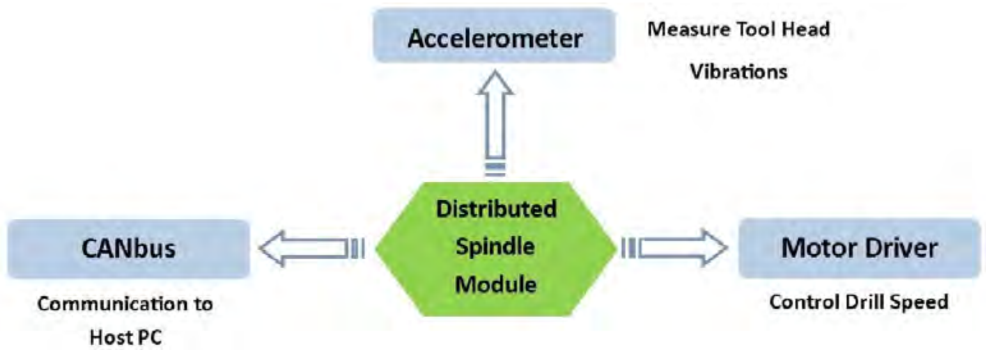


Figure 27: Diagram Illustrating the Tasks of the Distributed Spindle Module

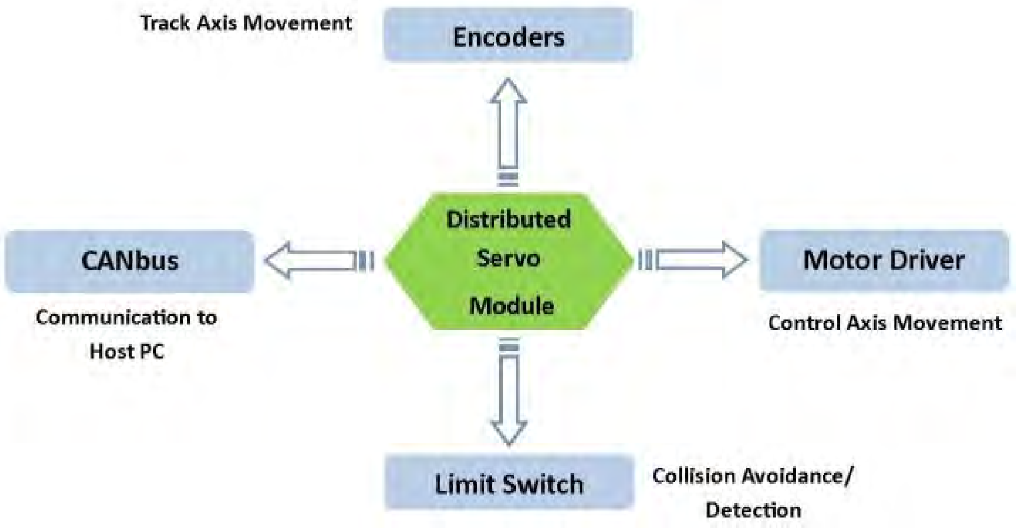


Figure 28: Diagram Illustrating the Tasks of the Distributed Servo Module

6.3 *Multiple Microcontroller Implementation*

Interoperability and openness are critical features for an OACS. The distributed modules were not only implemented on the FEZ Panda 2 boards but also on two additional development boards. The aim of this multiple platform implementation is to demonstrate that despite the varying differences of these three development platforms, with a little customization and circuit design, the distributed modules could be implemented on different platforms resulting in the same performance in linking to the host PC and control.

The chipKIT MAX32 shown in Figure 29 [34], is a high performance 80 MHz 32-bit microcontroller based on the PIC32MX795F512L, with the following key features [34]:

- Operating frequency: 80 MHz;
- I/O pins: 83 total;
- Analog inputs: 16;
- DC current per pin: 18mA;
- 2 CAN controllers.

Figure 29: chipKIT Max32

The Arduino UNO, shown in Figure 30 [35], is a 16 MHz 8-bit microcontroller based on ATmega328, with the following key features [35]:

- Digital I/O pins: 14 (6 PWM);
- Analog input Pins: 6;
- Clock speed: 16 MHz.

Figure 30: Arduino UNO

Table 3 summarises the key differences between the three chosen microcontrollers to implement the distributed modules on. As can be seen, the microcontrollers are not only different in their architectures and capabilities, but also their development environments and bootloaders.

Table 3: Comparison of Critical Features of the 3 Boards

	FEZ Panda 2 [33]	chipKIT MAX32 [34]	Arduino UNO [35]
Chipset	USBizi ARM7	PIC32	Atmel ATmega328
Operating Voltage	5 V	3.3 V	5 V
Operating Frequency	72 MHz	80 MHz	16 MHz
Digital I/O	54	83*	14
Analog Input	6	16	6
PWM Channels	6	6(Digital I/O)	6 (Digital I/O)
Can Interface	1x CAN controllers	2x CAN controllers	SPI via CAN-BUS Shield
Programming Environment	Visual C# with .NET Micro Framework	MPIDE	Arduino IDE
*3.3V output, but 5V tolerant for inputs			

6.4 Communications Network

The distributed control modules located at different points on the MRMT are addressed directly from the host PC. The host PC communicates with each module individually or to all the modules requesting data. Therefore, the communication interface had to satisfy the following criteria:

- Robust;
- Address and communicate with each module individually;
- Scalable and extendable – add, swap or remove modules with ease;
- Fast - handle real time control data;
- Sufficient Bandwidth – manage downloads and key control parameter transfers to modules;
- Reliable -minimize data packet losses;
- Interoperable – Ability to maintain communication links regardless of type of interfacing modules;
- Diagnosable – built in error checking that assists in runtime debugging.

Communication networks with multiple nodes can be linked using two approaches:

- Point to Point wiring from each node to the main node
  - E.g. RS232, RS485
- Bus network, wherein a bus is connected between two nodes and any other node can be connected between these without affecting the communication link.
  - E.g. Profibus, CAN bus, Foundation Fieldbus

Figure 31 illustrates the connections and complexity between these two connection approaches. From Figure 31 and the requirements for the communication interface, it is evident that the traditional approach of point to point communication links between all modules and the host PC would not be practical or suitable. The point to point wiring approach would not satisfy the aims of scalability, interoperability or a plug and play system with seamless integration. It would add to the complexity of the electronic circuit design, thereby adding more points of failure and decreasing the diagnosability by increasing the complexity of the network. In a scalable network which is likely to undergo reconfiguration, such a wiring system is not ideal.

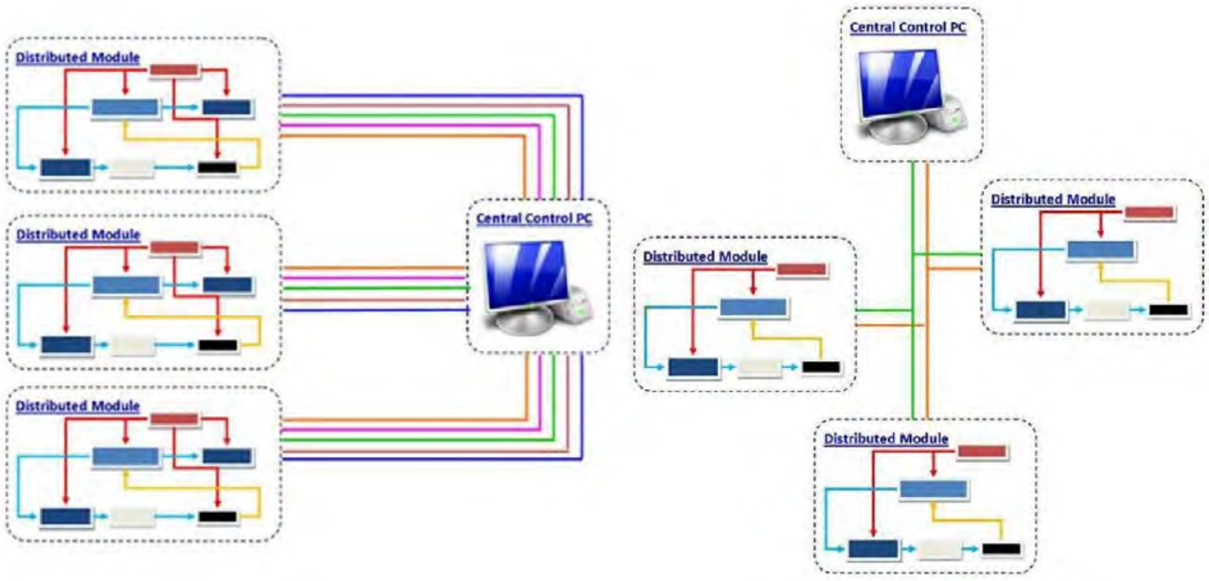


Figure 31: Comparison Between the Wiring for a Multi-wire vs CAN bus Network

The bus network approach reduces the complexity of the connections between nodes while adding additional circuitry at each node to handle the data transfer. The reduction in connections decreases the points of failure thereby increasing the diagnosability of the system.

Profibus, Foundation Fieldbus, Modbus and CAN bus are common industrially proven bus network interfaces and were all compared so that the best solution could be chosen for the OACS.

Profibus and Foundation Fieldbus originated from the same working group, and therefore have many commonalities, the key difference being the location of the interface between the field instrument and the control system [36]. Profibus network control systems maintain control on the control system, whereas Foundation fieldbus actually moves the control function to the field instrument.

Modbus is an open serial communication protocol that was developed in the 1979 for PLC [37]. Modbus is commonly used to link instrumentation and control devices, or in data gathering applications to transmit data back to the main controller. A Modbus network has one master device and up to 247 slaves on one network, with each having a unique address.

The Controller Area Network (CAN) bus communication protocol was developed by Bosch in the 1980's primarily for the automotive industry, but since then, it has been used and proven in many industrial applications [38]. Just like Profibus, CAN bus maintains control of the control system and links the control system to the module. A comparison of essential characteristics networks architectures is presented in Table 4.

Table 4: Comparison Between Profibus, Foundation Field Bus, Modbus and CAN bus

	Profibus [36]	**Foundation Field Bus [36]**	**CAN bus [39]**	**Modbus [37]**
Speed	400ms (loop cycle time with 24 nodes)	400ms (loop cycle time)	Loop cycle time <120µSec (at 1MBPS)	Serial Baud Rates
Nodes	24	12	127	247
Typical Applications	Field instruments Process industry	Field instruments Process industry	Automotive and Aerospace industries	PLC's and SCADA's

Profibus and Foundation Fieldbus are primarily used in process industries, therefore wired loops need to be Intrinsically Safe (IS), where IS is defined as a restriction of electrical energy as a source for ignition [40]. The associated linking modules require advanced electronic circuitry to limit the power in the loops and maintain a reliable communication network. This additional requirement to makes the loops IS safe adds a significant additional cost to the network. A new Profibus network costs approximately $13,500 for a 100 loop network and even more for a Foundation field bus network [36].

Unlike Profibus and Foundation Fieldbus, Modbus and CAN bus are used in the automotive industry and industrial industries. Since CAN bus and Modbus are two wire serial bus protocols which require a standard 2 wire serial cable to connect two capable nodes, Modbus and CAN bus do not bear the same cost for the associated electronics and loop wiring as the IS rated Profibus and Foundation Fieldbus communication networks and protocols.

The open Modbus standard makes the Modbus protocol extremely desirable and relevant for the OACS. However the limitation with Modbus comes with the dedicated master and slave nodes and each node requiring a unique address. As mentioned in Chapter 5 the OACS not only communicates with each node individually, but also to all nodes during bus scans. Due to this requirement, the Modbus communication protocol is not suitable.

CAN bus differs fundamentally from Profibus, Foundation Fieldbus and Modbus in that each node does not necessarily need to have an address, but rather each message has a unique identifier [38]. Therefore, all nodes receive all transmitted messages and can choose to accept or reject the messages.

Due to the reduced complexity, simplified communication protocol, lower cost factor as well as the message identifiers as opposed to dedicated node identifiers, the CAN bus protocol was chosen as the best solution for the MRMT.

6.5 CAN bus

CAN bus is widely used in the automotive and aerospace industries making it a robust, tried and tested communications network [41]. CAN bus has built-in collision avoidance and rescheduling of messages in the case of nodes wishing to transmit at the same time [38], a valuable feature to have available in a multi node network.

CAN bus has built-in collision avoidance and rescheduling and by monitoring the bus it checks if two nodes attempt to transmit at the same time. If this occurs, the system commands the higher priority node to transmit, and reschedules the second message [38]. Similarly when a collision occurs on the bus, the nodes check the priority of the messages, and the message with the highest priority, or lowest message identifier is transmitted immediately, while the lower priority message is delayed and rescheduled [38].

Hence the CAN bus network architecture enables the communication interface to be scalable, extendable and interoperable allowing up to 127 nodes which can be seamlessly integrated with its plug and play interface. The communication protocols allow transmissions to an individual module or to all modules at once. In addition, the built-in error checking and collision avoidance features increases the reliability of the system. Due to the simplicity of connections the system has a high inherent diagnosability.

The CAN bus communication protocol assigns a message identifier to each message and all nodes receive all transmitted messages. Each node chooses to accept or reject the received transmission based on this identifier. Due to the reconfigurability, un-determined structure of the MRMT such a feature further assists the system by allowing system modifications changes and allowing the system to adapt to manage the physical changes. ID of 0x00 is reserved as a general message and all nodes are programmed to accept this message and reply with their unique ID and type of distributed module. Each distributed module is assigned a unique ID or address on the network and in the case of a new module being added to the system, the distributed controller can be assigned a unique ID or address. When a new node is connected to the bus, any of the other modules is able to access it via its unique ID or address. Furthermore, since the host PC runs a scan on the bus on start-up, there is no predetermined setup and configuration of the system. This configuration allows the system to be truly reconfigurable, scalable and extendable.

The OACS has been designed with the identifiers shown on Table 5 pre-programmed on to the respective nodes. Any new module added onto the system can be assigned with unreserved IDs 0x06 and greater. A sample list of acceptable message transmissions is detailed in Section 6.7.

Table 5: Unique Node Addresses/Identifiers

ID	Node	Details
0x00	All	General call to all modules
0x01	Host PC	
0x02	Translational Module 1	X Axis
0x03	Rotary Module 1	A Axis
0x04	Translational Module 2	Z Axis
0x05	Spindle Module	Drill

6.6 CAN bus Hardware

The PC is connected to the bus network via a PCAN PCI card from Peak Systems, the card uses a the 9 pole SUB-D connecter to output the CANH and CANL bus lines that connect to the nodes. The pin configuration of the 9 pole SUB-D connector is shown in Figure 32 [42] and the PCI card in Figure 33 [42].

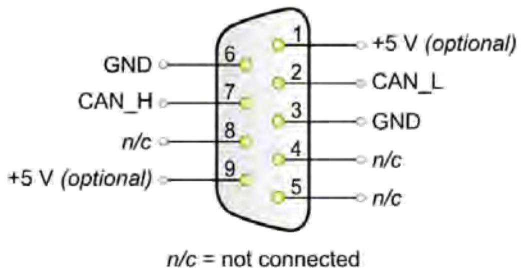


Figure 32: Pin Out Configuration of a 9 pole SUB-D Connecter

Figure 33: Peak Systems PCAN PCI Card

Figure 34 illustrates how the different nodes are connected together along the bus, and the bus termination resistors required ensuring that the transmissions do not reflect back on to the bus.

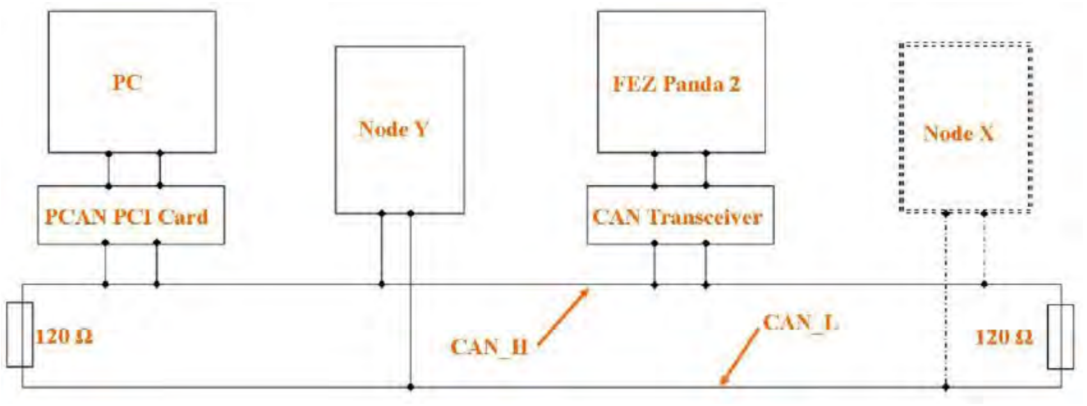


Figure 34: Illustration of CAN bus Network and Node Connections

The two bus lines, CANH and CANL, from the PCAN PCI card operate at 5V, whereas on the FEZ Panda 2, these two ports operate at 3.3V. Therefore in order to communicate over the bus lines conversion transceivers were required.

The 3.3V Texas Instruments SN65HVD231D CAN transceivers with a high input impedance that can connect up to 210 nodes on the bus was used [43]. The transceivers convert the higher voltages on the bus lines to TTL voltage levels that are compatible with the FEZ Panda 2.

The chipKIT based on the PIC32 microcontroller has an integrated CAN bus communication circuitry and features, but as with the FEZ Panda 2, voltage level conversion is required. The chipKIT network shield depicted in Figure 35 [44] has the following key CAN bus related features [44]:

- Two MCP2551 CAN Transceivers
- Two 12-pin header connectors for CAN
- 32.768 KHz Oscillator for Real time clock

Figure 35: chipKIT Network Shield

Unlike the Fez Panda 2 and the chipKIT, the Arduino UNO doesn't have built in CAN functionality. However the CAN bus shield shown in Figure 36 [45], extends this for the Arduino Uno and has following key features [45]:

- CAN v2.0B up to 1 Mb/s
- High speed SPI Interface (10 MHz)
- Standard and extended data and remote frames
- CAN connection via standard 9-way sub-D connector

With the addition of a CAN bus shield which is based Microchip MCP2515 CAN controller and the MCP2551 CAN transceiver [45] , the CAN bus shield interfaces the Arduino UNO via high speed SPI communication and converts this to the CAN communication protocols, thereby giving the Arduino UNO full CAN bus capabilities.

Figure 36: CAN-BUS Shield

6.7 Message Packet Formatting

Each node has a unique address and when a message is transmitted from the host PC, the address of the receiving node is assigned as the message identifier. The fields of a message are shown in Figure 37.

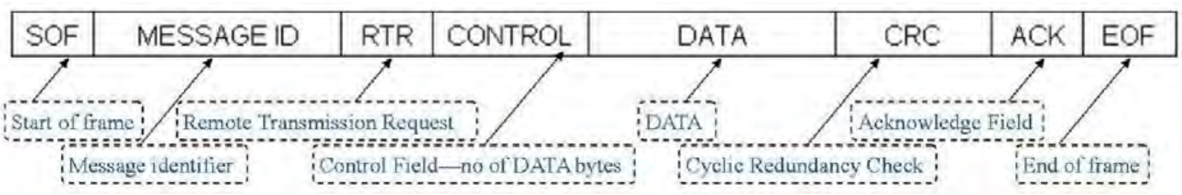


Figure 37: CAN bus Message Fields

The start of frame and the end of frame characters in the message packet are automatically generated. The Data, Message ID, and control field are generated depending on the type of message packet being transmitted. The control field sets the number of data bytes with the default set-up being 8 data bytes.

The host PC initially calls for data from the distributed modules and later on during operation it periodically sends and receives feedback data from the modules. Once this data has been processed and updated movement and control requirements are generated. These requirements are sent out to the respective distributed module.

The data calls from the host PC can either be a general call to all modules or a specific call to a module. A general call is used initially during a bus scan to detect which distributed modules are active and connected to the system. Thereafter during setup and configuration, the host PC asks each distributed module for specific data about its operation and limits.

A sample set of transmitted message packets from the host PC to the distributed modules that has been developed is shown in Table 6.

Table 6: Table Showing Packet Message Formats for Transmissions from Host PC

Message Type	TX MSG Address	TX data	Request
1 – General Call to all Modules	0x01 – reserved address for a general call	1 0 0 0 0 0 0 0	Modules to reply with their ID's and a code for the type of module (e.g Servo/spindle)
2 – General Call to all Modules	0x01 – reserved address for a general call	2 0 0 0 0 0 0 0	Modules to reply with their ID's and a the type of module (e.g Linear Axis X)
3 – Move Command (Start)	0x** - distributed module ID	3 1 X Y 0 0 0 0	Module to Start - X is the value to move (Y =1 CW, Y = 0 CCW)*
4 – Move Command (Stop)	0x** - distributed module ID	3 0 X Y 0 0 0 0	Module to Stop - X is the value to move (Y =1 CW, Y = 0 CCW)*
5 – Move Command (Switch Direction)	0x** - distributed module ID	3 2 0 0 0 0 0 0	Module to Switch Direction
*CW – Clock Wise Rotation, CCW, Counter Clock Wise Rotation ** - Specific distributed module ID			

Based on the messages received and the message identifier, samples of the format that the distributed module uses to process the received data using is shown in Table 7.

Table 7: Table Showing Packet Message Formats for Transmissions from Distributed Modules

Message Type	MSG Address	TX data
Reply to General Call 1 - Modules replying with their ID's and a code for the type of module (e.g Servo/spindle)	0x00 – addressed to the PC	1 boardID d d d d d d (generic format) e.g. 1 # S e r v M * e.g. 1 # S p i n M* * 1-reply to call 1, # is the board ID
Reply to General Call 2 - Modules to reply with their ID's and a the type of module (e.g Linear Axis X)	0x00 – addressed to the PC	2 boardID d d d d d d (generic Format) e.g. 2 # L i n r X a* e.g. 2 # D r i l l T * *2-reply to call 2, # is the board ID

Reply to Move command 3 or 4– An acknowledgement from the distributed module signalling the command has been carried out successfully.	0x00 – addressed to the PC	4 boardID X Axis . . . .
		4 boardID X Axis . . . .
		X = 0 – Start Reply
		X = 3 – Stop Reply

Periodic transmission from distributed modules to PC indicating axis position(from encoders)	0x00	3 BoardID X Axis Y . . . X = 1 Axis Position, Y is the actual position of the axis

The host PC sequentially decodes the received data from the distributed modules and stores it under the corresponding classes, an explanation of the decoding and storage of data is covered in Chapter 7.

6.8 Spindle and Axis Speed Control

The 15 A Pololu H-Bridge motor drivers depicted in Figure 38 [46] were used to control the motors of the servo axis and spindle tool head modules for digital spend and position control. The MOSFET H-Bridge motor drivers are capable of bidirectional speed control of a single high voltage motor delivering up to 15 A continuous current [46].

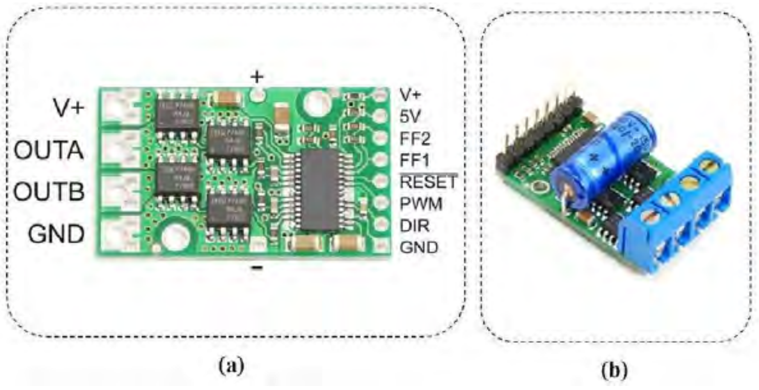


Figure 38: (a) Pin Out Configuration of Motor Driver (b) Assembled Motor driver

The H-Bridge motor drivers require two input signals, a Pulse Width Modulation (PWM) signal and a direction signal [46]. These two signals are provided by the distributed modules. Table 8 summarises the operation and controls required for the drivers.

Table 8: Control Signals and Operation for Motor Drivers

PWM	Direction	OUT A	OUT B	Operation
H	L	L	H	Forwards
H	H	H	L	Backwards
L	-	L	L	Stop

The chipKIT microcontroller is only capable of outputting 3.3V on a pin. As a result voltage level conversion was required to ensure the motors could be driven with a consistent 5V PWM source.

6.9 Motor Noise Cancellation

The high voltages and currents that DC motors require for operation can result in Electromagnetic Interference (EI) [47]. If this EI, more commonly known as noise, is close enough to low voltage supplies or signal wires it can affect the stability as well as the values being transmitted on these lines.

Therefore it is necessary to filter these high frequency noise signals to ensure minimal interference. Several methods to filter out the noise exist, varying in complexity and cost. Capacitors, ferrite rings, power supply separation and twisted pairs of wires were all implemented to ensure that no interference resulted between circuits.

Capacitors can hold the voltage on the line constant for a short period of time due to the voltage stored in the capacitor. If a capacitor is on a voltage line and a voltage spike occurs as a result of the motor noise, the capacitors hold the line at a steady state voltage for a short period of time, thereby cancelling the effect of the short spike. The period of time is dependent on the value of the capacitor.

In order to filter out the motor noise from the DC motors, 3 capacitors were used in a parallel and series combination as shown in Figure 39. The capacitors C1 and C2 connected between the motor terminals and the motor casing has the effect of shorting the casing and the terminals together, thereby making the casing of the motor a shield and assisting in reducing the radiated noise and the electromagnetic interference with surrounding circuits [47]. The single capacitor C3 connected between the motor terminals acts as short circuit for the high frequency nose. This connection does not affect the power to the motor but reduces the conduction of noise along the motor wiring [47].

This use of capacitive filtering reduces high frequency interference. To complement the capacitors, a low frequency conducting method was introduced. The motor wiring was looped several times through a ferrite ring, this looped set of wires only conduct low frequency signals such as the power to the motor and assist in further reducing out high frequency signals [47].

On start-up motors draw larger currents than during normal operation. If the power supply to the motors and the other electronics are the same, due to the current demands of the motors on start-up, the voltage and current supplied to the other circuitry can be affected. It is therefore best to completely isolate and separate the power supplies for the motors and the other electronic circuitry. The power distribution overview and design is covered in detail in section 6.13.

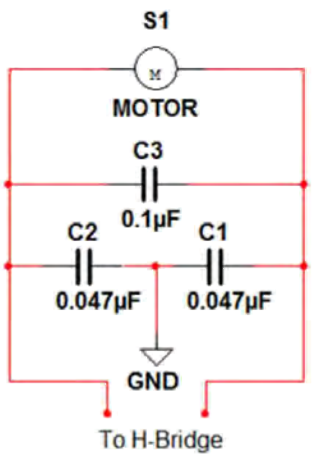


Figure 39: Motor Terminals With Added Capacitors to Filter Out Noise

In addition to the filtering and separation methods, a simple yet effective method of countering electromagnetic interference is to use twisted pairs of wires to cancel the induced currents. When two wires travel alongside each other they can have an induced current created in them depending on the distance between the wires and the source of the electromagnetic noise source [47]. The distance between the wires creates a potential difference between the wires, thereby resulting in a noise current induced in the wires. If the wires are twisted, they alternate in being closer to the source of electromagnetic noise and the potential differences in one section is of opposite polarity to the next section, which results in a cancellation of noise along the whole wire [47].

6.10 Collision Detection

The translational modules which are controlled by the axis motors operate in a limited work space and it is necessary to detect if the translational modules are approaching the end of the workspaces to ensure no damage to components occur as a result of collisions. Limit switches, as shown in Figure 40 [48], were used to detect the movement of a translational module at the ends of each axis.

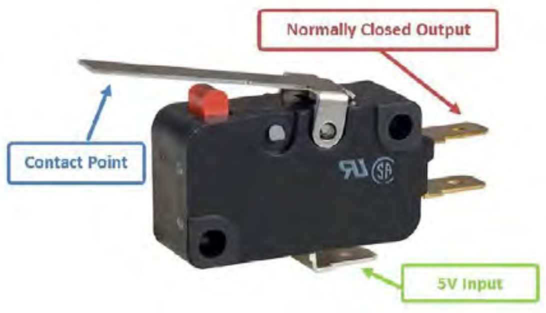


Figure 40: Limit Switch

The number of limit switches surrounding a translational module differs depending on the type of module. Figure 41 [29], further illustrates how two limit switches are placed at either end of a liner translational axis to signal the end of movement for the motion module.

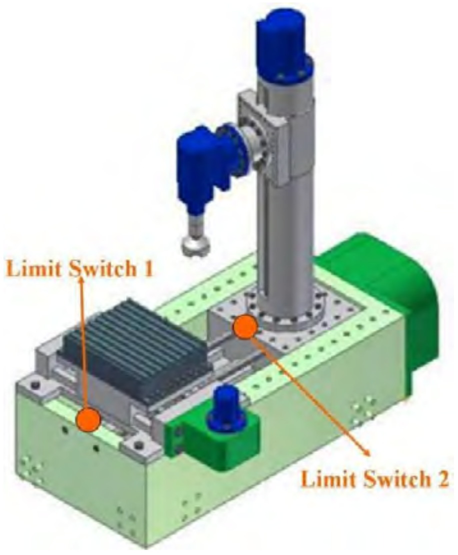


Figure 41: Illustration of Limit Switches on the MRMT

The limit switches were connected in a normally-closed configuration. When contact occurs as a result of the moving translational module making contact with the limit switch, the limit switch circuitry would open and the signal would trigger an interrupt on the distributed modules. By making the limit switches “normally closed”, it ensures fails safe design as if the circuit breaks unexpectedly, the interrupts would trigger and stop operation. If the design followed a normally-open configuration, and if the circuit breaks with the MRMT in operation and when it approaches the axis limit, no signal would trigger and this would lead to damage of the MRMT.

The output signal from the limit switch, which was initially a steady 5V signal, would fall to 0V. This signal is connected to an external interrupt pins on the corresponding distributed module. When the translational axis makes contact with the limit switches, the interrupt routines are triggered, and these routines stop the PWM output signals to the motor drivers, in turn stopping the motor, ensuring no damage to the translational modules.

A common problem with electro-mechanical switches is switch debouncing. When a switch is pressed, there is a chance that the output may swing from rail to rail before finally settling at a steady output. During this interval, due to the speed of the microcontrollers, the microcontrollers detects as if the switch is pressed several times. To counter these erroneous detections, a software and hardware solution was implemented.

The software solution on the Fez Panda 2 makes use of a glitch filter provided by Microsoft’s .NET Micro Framework. The glitch filter checks the time between the interrupts coming in from the switch and it rejects interrupts that are within a time set by the user [49].The glitch filter was used on all the interrupts pins for the limit switches and is activated by the following lines of code:

```
TimeSpan ts = new TimeSpan(0, 0, 0, 0, 100);//100ms Glitch filter
Microsoft.SPOT.Hardware.Cpu.GlitchFilterTime = ts;
```

Similarly, on the chipKIT and the Arduino UNO, software routines checks the time between the interrupts and only triggers the interrupt service routine to stop the motors when the time between corresponding interrupts is greater than 100 ms.

In hardware a Schmitt trigger circuit, also known as an inverting buffer or a not gate, was added between the output of the limit switch and the microcontroller. The Schmitt trigger circuit was implemented using a 555 timer with the following pin configuration in Figure 42 [50].

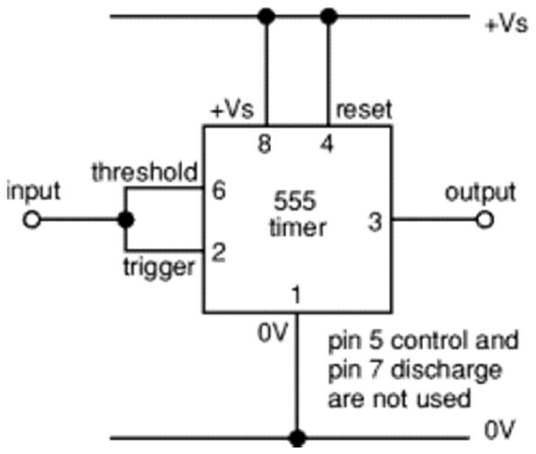


Figure 42: Schmitt Trigger Connection Diagram for the 555 Timer

Table 9: Schmitt Trigger Detector Input Conditions and Outputs

Input	Condition	Output
LOW	< 1/3 Vs	High
HIGH	>2/3 Vs	LOW
LOW/HIGH	1/3Vs < X < 2/3 Vs	Same as Input

The Schmitt trigger detector output is determined from Figure 9. From Table 9, when the output is changed to logic HIGH or LOW, it must be changed by 2/3Vs to switch to its previous state, thereby giving the circuit a high immunity to noise [50].

Both the software and hardware design to counter switch debouncing ensure that no false triggers stop the motors. Only when the collision avoidance limit switches are activated is the routine to stop the motors called.

6.11 Position Feedback Encoders

Three channel HEDS 5540 optical encoders were used to track the movement of the motors. The encoders are capable of a resolution of 1024 counts per revolution or less depending on the configuration used [51].

These optical encoders contain a lensed LED source and built-in detection and output circuitry. A block diagram of the circuitry is shown in Figure 43(a) [51]. As the internal code wheel rotates between the LED source and the detector circuitry, two quadrature square wave output signals are generated. In addition to the two quadrature outputs, a third channel on the encoders, the index channel is also generated.

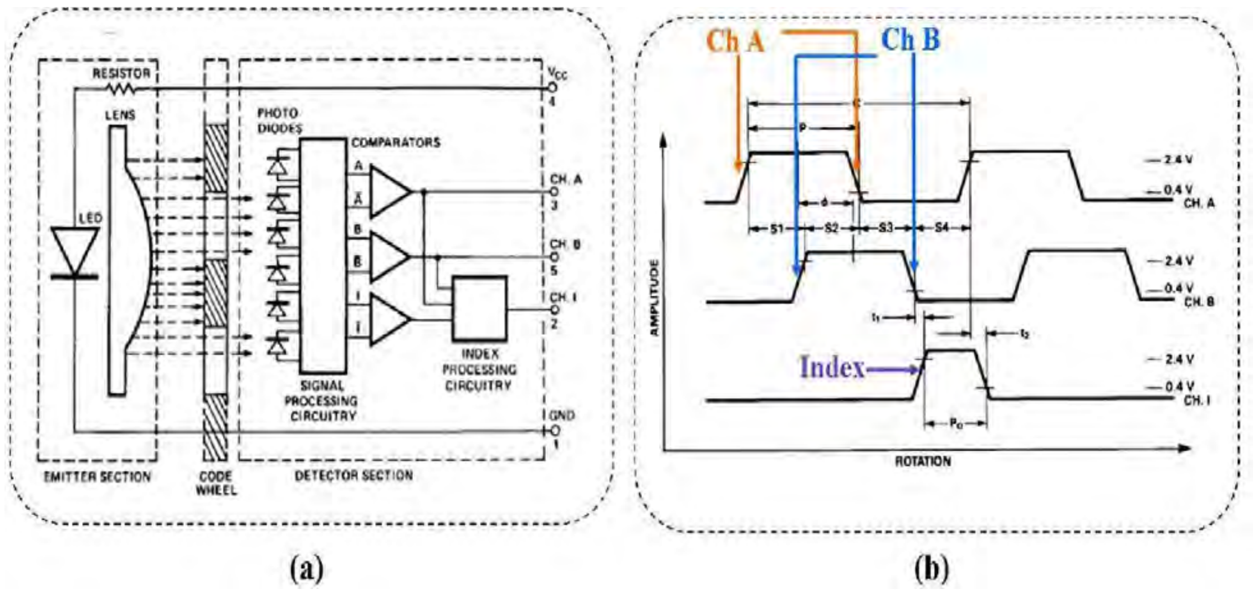


Figure 43: (a) Block Diagram for the Optical Encoder (b) Output Signals From the Encoder

The index pulse is generated once per rotation of the code wheel and the channel A and channel B pulses are generated in a quadrature format as shown in Figure 43(b). The 4 pulses generated by Channels A and B are each generated 256 times in every rotation adding up to a count of 1024 per revolution. The direction of rotation can be determined by analysis of the quadrature waveforms in Figure 43(b) [51].

The output signals from the three channels are connected to the external interrupt pins on the distributed modules. When the interrupts trigger, a counter is incremented or decremented depending on the result of this check. The incrementing and decrementing of the counter imply a movement of the axis which has a zero set point located at the midpoint of the axis.

The encoder output also followed a fail-safe design principle to ensure no false triggering occurs. The three input channels A, B and Index were connected to the distributed module interrupt pins via 2.7 kΩ pull-up resistors which sets the threshold voltage for triggering.

6.12 Vibration Sensor: Accelerometer

The MRMT would be set up and configured to operate a drill, being a CNC machine, it is critical to assess the performance of the drilling action of the MRMT. In order to test and monitor the accuracy of the stability of the drill module on the MRMT, a 3 axis accelerometer is installed on the drill module.

The chosen accelerometer was the ADXL335 from Analog devices, a 3 axis and 3G accelerometer [52]. The ADXL335 measures static acceleration of gravity in tilt applications in addition to dynamic acceleration as a result of movement and vibration [52]. The drill head module would be moving and to monitor the vibrations, the ADXL335 breakout board, shown in Figure 44 [53], was used.

Figure 44: SparkFun ADXL335 Breakout Board Triple Axis Accelerometer

6.13 Power Distribution Network

The power distribution network is powered from 230V AC 50Hz power source and uses three AC to DC power supply units to power up the motors, distributed modules and all associated sensor circuits. The three AC to DC power supplies are:

- One 12 V DC 29 A unit;
- One 5 V DC 7 A unit;
- One 3.3 V DC 6 A unit.

The actual power usage of the system is highly dependent on the number of mechanical modules and in turn associated electronics, distributed module and sensors which are active on the system. Therefore, the current power distributed network has been designed and implemented to ensure that the system has the ability to be scalable and extendable in terms of adding additional mechanical modules on to the MRMT and ensuring system functionality without the need to modify other aspects of the overall system. Figure 45 summarises the power distribution network.

The dotted lines indicate the connections that will be wired in order to add in additional modules. An analysis of the power distribution system in terms of current consumption and the limitation of number of modules with the current power supplies are discussed in Chapter 9.

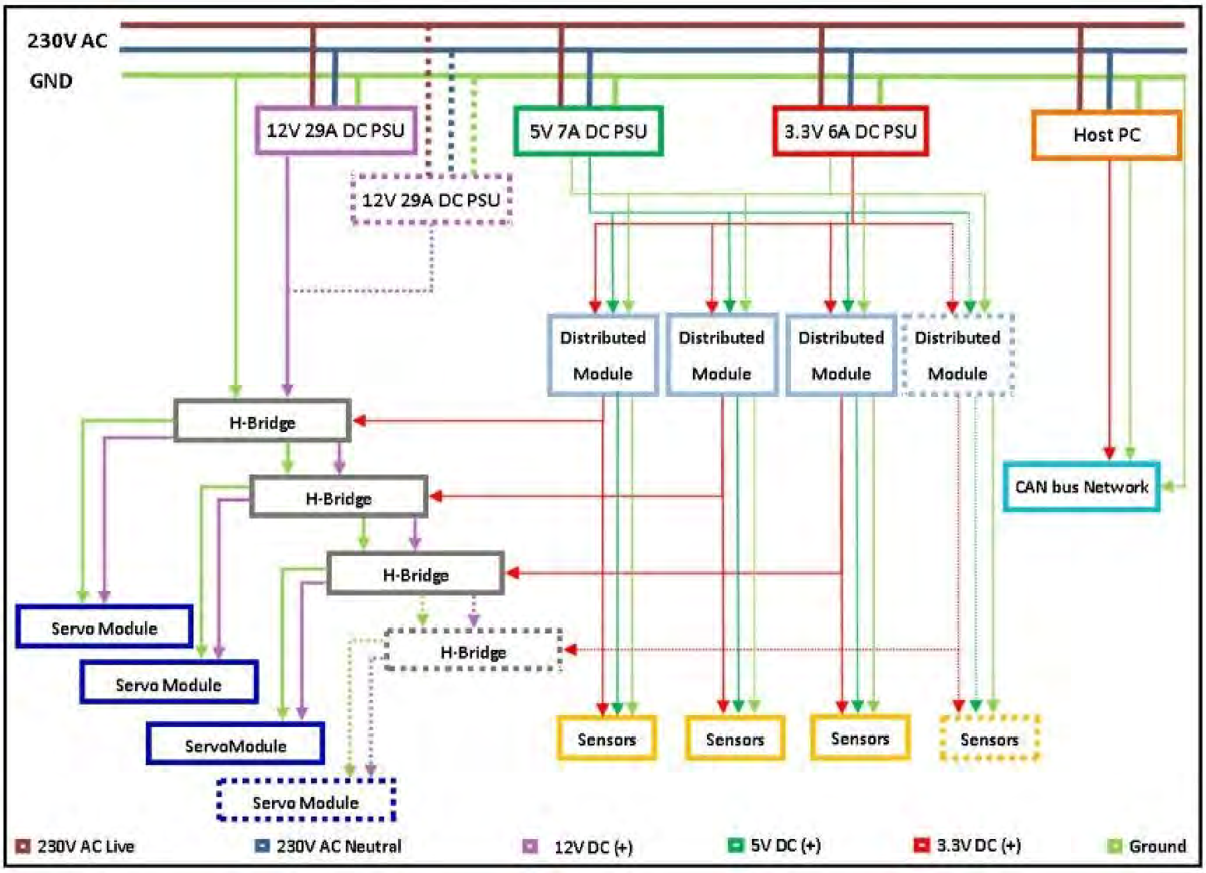


Figure 45: Power Distribution Network

6.14. Chapter Summary

Chapter 6 has presented the core aspects of the electronic sub systems of the research and design. The multiple microcontroller implementations of the distributed modules has been presented followed by a discussion presenting the communication interface, the sensors and the design methods to overcome false triggering and noise interference. Finally a high level overview of the power distribution network is presented indicating the key power distribution channels across the whole electronic subsystem.

7. Control Algorithms for Open Architecture Control System

The OACS has several algorithms coded in to simplify the user interface, and to give the end user access to: customizable, reconfigurable and control features. Five algorithms have been added into the OACS. The algorithms and their corresponding purposes/functions/tasks are as follows:

- Kinematic Viability;
  - The purpose of which is to determine if the mechanical configuration of the MRMT is kinematically viable
- Control Theory;
  - PI or PID control
  - Tuning of PI or PID control loops
    - Trial and Error
    - Ziegler Nichols Method
- Text Interpretation and User program Validation;
  - Interpretation of the user entered program
  - Ensuring the user entered program is executable on the current MRMT
- Interpolation;
  - Determining the movement requirements of each axis and relative speeds
- Acceleration and deceleration Control;
  - To prevent mechanics shock and destination point overshoot

Figure 46 summarises the tasks that the user follows in order to setup and configure the MRMT. In addition, the aforementioned algorithms are also illustrated in order of occurrence and are indicative of when each algorithm is run.

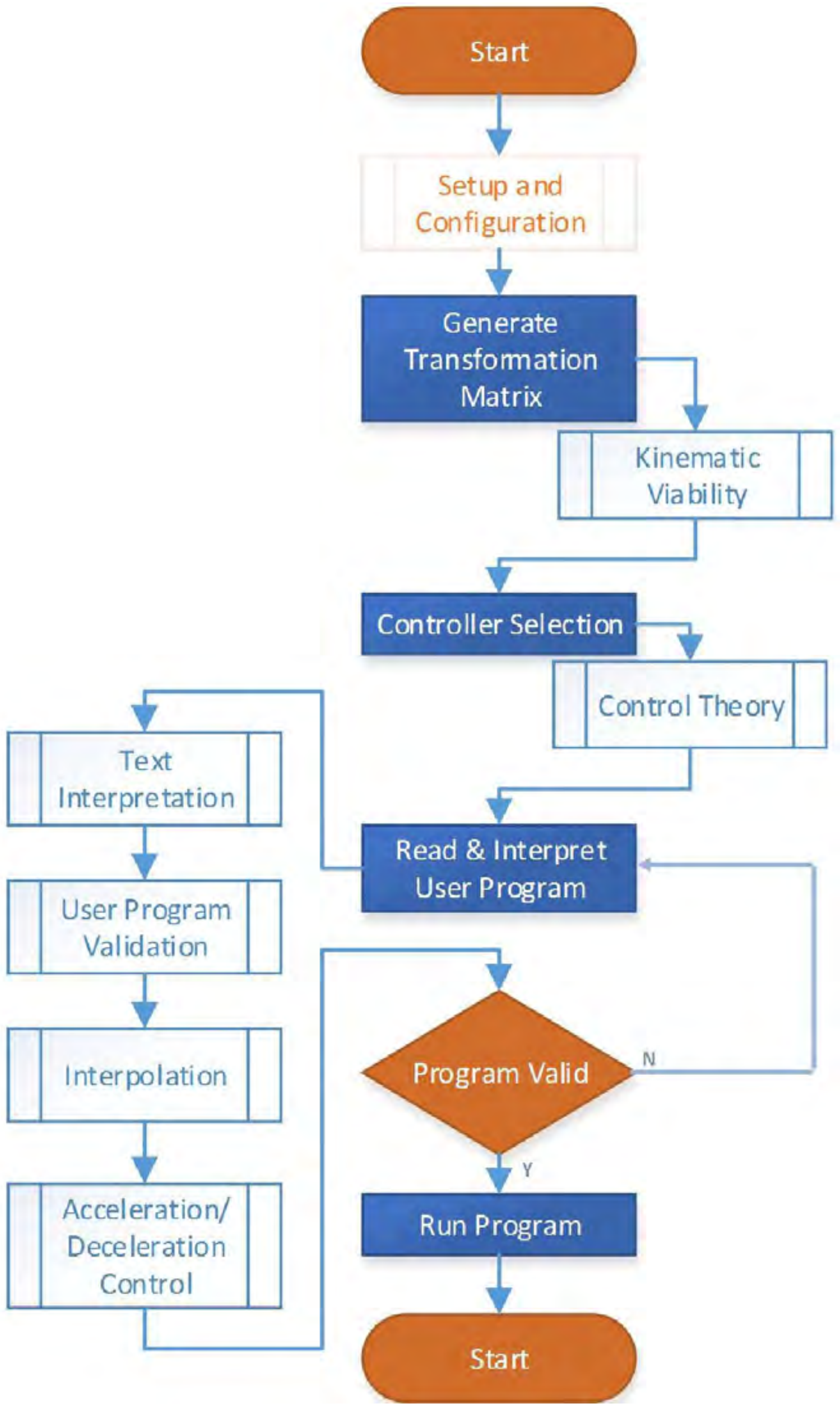


Figure 46: Flowchart Illustrating Embedded Algorithms

7.1. Kinematic Modelling

The operation of the RMT is defined by the kinematic motion between the tool module and the work piece. The kinematic motion describes the relationship between the joint position and the end effector position with reference to an absolute reference. In order to determine if the chosen set of mechanical modules on the MRMT combined to form a workable solution, the kinematic solution of the module combination needed to be evaluated [54].

The mechanics of a MRMT can be described as a kinematic chain of rigid bodies connected by joints. Any motion of the MRMT can be described by the computation of the individual motions of each link in reference to the previous link. There is a kinematic relationship between two links, a translational and rotary relationship [55].

At the $\boldsymbol{i^{th}}$ joint on the MRMT, let $\boldsymbol{q_i}$ denote the joint variable at $\boldsymbol{i}$. $\boldsymbol{q_i}$ can either be an angle or displacement depending on the type of joint, and is represented by equation 7.1 [56]:

$$\boldsymbol{q_i} = \begin{cases} \boldsymbol{\theta_i} \\ \boldsymbol{d_i} \end{cases} \tag{7.1}$$

If the matrix ***M***, a Homogenous Transformation Matrix (HTM), represents the position and orientation at $\boldsymbol{i}$ with respect to $\boldsymbol{i-1}$ for an axis. ***M*** is dynamic and can change as the MRMT undergoes reconfiguration. For the MRMT, each axis is a function of only a single degree of motion, therefore:

$$\boldsymbol{M_i} = \boldsymbol{M_i}\,(\boldsymbol{q_i}) \tag{7.2}$$

Similarly the HTM that expresses the overall configuration of the MRMT, from position $\boldsymbol{j}$ with respect to position and orientation $\boldsymbol{i}$, is represented by equation 7.3 [56]:

$$\boldsymbol{T_j^i} = \boldsymbol{M_{i+1}} \dots \boldsymbol{M_j} \tag{7.3}$$

To simplify this, if the rotary motion can be defined as $\boldsymbol{R_n^0}$, a 3x3 rotation matrix and the translational motion by $\boldsymbol{O_n^0}$, a 3x1 matrix [56], then equation 7.3 can be manipulated to equation 7.4:

$$\boldsymbol{T} = \boldsymbol{H} = \begin{bmatrix} \boldsymbol{R_n^0} & \boldsymbol{O_n^0} \\ \boldsymbol{0} & \boldsymbol{1} \end{bmatrix} \tag{7.4}$$

Since each axis of the MRMT is limited to a single DOF, equation 7.4 can be represented as equation 7.5:

$$\boldsymbol{H} = \boldsymbol{T_n^0} = \boldsymbol{M_1}\,(\boldsymbol{q_1}) \dots \boldsymbol{M_n}\,(\boldsymbol{q_n}) \tag{7.5}$$

Similarly, equation 7.6 can represent the task transformational matrix:

$$\boldsymbol{T_j^i} = \boldsymbol{M_{i+1}} \dots \boldsymbol{M_j} = \begin{bmatrix} \boldsymbol{R_j^i} & \boldsymbol{O_j^i} \\ \boldsymbol{0} & \boldsymbol{1} \end{bmatrix} \tag{7.6}$$

To simplify the kinematic representation, the Denavit Hartenberg representation can be used. The D-H convention represents each HTM $\boldsymbol{M_i}$ by a product of four basic transformations as shown in equation 7.7 and equation 7.8. The four parameters $\boldsymbol{\theta_i}\, \boldsymbol{d_i}\, \boldsymbol{a_i}$ and $\boldsymbol{\alpha_i}$ derived from Figure 47 [56] describe a relationship between movements [54].

$$\boldsymbol{M_i} = \boldsymbol{R_{z,\theta_i}}\, \boldsymbol{Trans_{z,d_i}}\, \boldsymbol{Trans_{x,a_i}}\, \boldsymbol{R_{x,\alpha_i}} \tag{7.7}$$

$$= \begin{bmatrix} c_{\theta_i} & -s_{\theta_i} & 0 & 0 \\ s_{\theta_i} & c_{\theta_i} & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & a_i \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & c_{\alpha_i} & -s_{\alpha_i} & 0 \\ 0 & s_{\alpha_i} & c_{\alpha_i} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\boldsymbol{M_i} = \begin{bmatrix} \boldsymbol{c_{\theta_i}} & \boldsymbol{-s_{\theta_i}c_{\alpha_i}} & \boldsymbol{s_{\theta_i}s_{\alpha_i}} & \boldsymbol{a_ic_{\theta_i}} \\ \boldsymbol{s_{\theta_i}} & \boldsymbol{c_{\theta_i}c_{\alpha_i}} & \boldsymbol{-c_{\theta_i}s_{\alpha_i}} & \boldsymbol{a_is_{\theta_i}} \\ \boldsymbol{0} & \boldsymbol{s_{\alpha_i}} & \boldsymbol{c_{\alpha_i}} & \boldsymbol{d_i} \\ \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{1} \end{bmatrix} \tag{7.8}$$

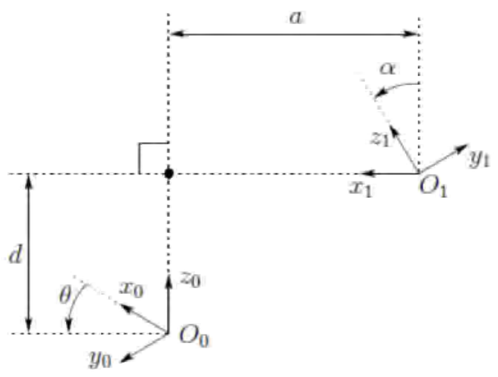


Figure 47: Coordinates References Between Movements

Since each axis on the MRMT is a function of a single degree of freedom, the matrix $\boldsymbol{M_i}$ can be simplified by using Euler angles. The Euler representation simplifies the description of motion by describing the orientation of a coordinate relative to another. For the MRMT, the rotations are fixed along the principle axes X, Y and Z. We can define this rotation at orientation at $\boldsymbol{i}$ with respect to $\boldsymbol{i-1}$ as follows in equation 7.9 [57]:

$$R_{zyx} = \begin{bmatrix} \cos\alpha\cos\beta & \cos\alpha\sin\beta\sin\gamma - \sin\alpha\cos\gamma & \cos\alpha\sin\beta\cos\gamma + \sin\alpha\sin\gamma \\ \sin\alpha\cos\beta & \sin\alpha\sin\beta\sin\gamma + \cos\alpha\cos\gamma & \sin\alpha\sin\beta\cos\gamma - \cos\alpha\sin\gamma \\ -\sin\beta & \cos\beta\sin\gamma & \cos\beta\cos\gamma \end{bmatrix} \quad (7.9)$$

Using the rotation description and combining it with the D-H representation, the HTM of each axis $\boldsymbol{M_i}$ can be represented as 7.10 [56]:

$$^{i+1}_{\quad i}M_n = \begin{bmatrix} \cos\alpha\cos\beta & \cos\alpha\sin\beta\sin\gamma - \sin\alpha\cos\gamma & \cos\alpha\sin\beta\cos\gamma + \sin\alpha\sin\gamma & x \\ \sin\alpha\cos\beta & \sin\alpha\sin\beta\sin\gamma + \cos\alpha\cos\gamma & \sin\alpha\sin\beta\cos\gamma - \cos\alpha\sin\gamma & y \\ -\sin\beta & \cos\beta\sin\gamma & \cos\beta\cos\gamma & z \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (7.10)$$

When the MRMT is configured, the HTM of each axis module is downloaded from the distributed modules. Based on the user input, which requests the physical configuration of mechanical modules on the MRMT, the computation for the complete task transformation matrix is determined.

$$T_{Work\ \ Tool} = M_n \dots M_{n-1} M_1 \quad (7.11)$$

This forward kinematic model is determined by the multiplication of the individual HTM of each axis on the MRMT in an ordered manner. The task transformation matrix describes the position of the MRMT tool module relative to the global reference on the work holding module at the base of the MRMT.

Based on the derivation of the HTM of each axis, the HTM of each axis is presented in equations 7.12 to equation 7.16 using the offsets due to positioning of the modules on the MRMT:

$$^{i+1}_{\quad i}M_{X\,Axis} = \begin{bmatrix} 1 & 0 & 0 & x \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & -111 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (7.12)$$

$$ {}^{i+1}_{i}M_{\,Z\,Axis} = \begin{bmatrix} 1 & 0 & 0 & -54.5 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & z \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{7.13} $$

$$ {}^{i+1}_{i}M_{\,A\,Axis} = \begin{bmatrix} cos\alpha & -sin\alpha & 0 & 0 \\ sin\alpha & cos\alpha & 0 & 0 \\ 0 & 0 & 1 & -152 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{7.14} $$

$$ {}^{i+1}_{i}M_{\,Drill} = \begin{bmatrix} 1 & 0 & 0 & -7\ 5 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & -81.5 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{7.15} $$

$$ {}^{i+1}_{i}M_{\,Ext\,Arm} = \begin{bmatrix} 1 & 0 & 0 & -3\ 7\ 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{7.16} $$

7.2. Control Theory

The axis control modules for the MRMT can be categorized into three-tier architecture as depicted in the overview in Figure 48. The three tiers in this architecture are as follows:

- The adaptive control module and error compensation in the top layer;
- The interpolation module in the middle layer;
- The servo and spindle control modules in the bottom layer.

The adaptive control module controls the spindle speed and feed rates based on the user programmed values. The error compensation module uses the sensor feedback data and compares the expected machine position with the actual position. Based on this comparison, the error compensation module corrects any deviation in movement and operation by compensating the next set of values and data sent out to the distributed modules.

The spindle speed rates generated by the adaptive control module are sent directly from the top layer to the bottom layer and then to the spindle control module. The spindle control module in turn generates the movement commands, and transmits these commands to the spindle driver circuitry.

The updated and compensated movement instructions as well as the updated feedrate commands are sent form the top layer to the middle layer and then to the interpolation module. The interpolation routine determines the interpolated position commands and in-turn sends them down to the distributed modules at the bottom layer. The servo modules generate the movement commands and transmit these commands to the axis driver circuitry.

The servo and spindle control modules are capable of controlling machine tool or axes movement over a range of speeds. The spindle control module controls tool drilling or cutting speeds to ensure correct machining operation, and the servo control modules controls axes movement for best accuracy. To achieve such control, a closed loop control system is required. In a closed loop control system, external sensors on the axes monitor the axes movement and constantly provide feedback data to the control modules.

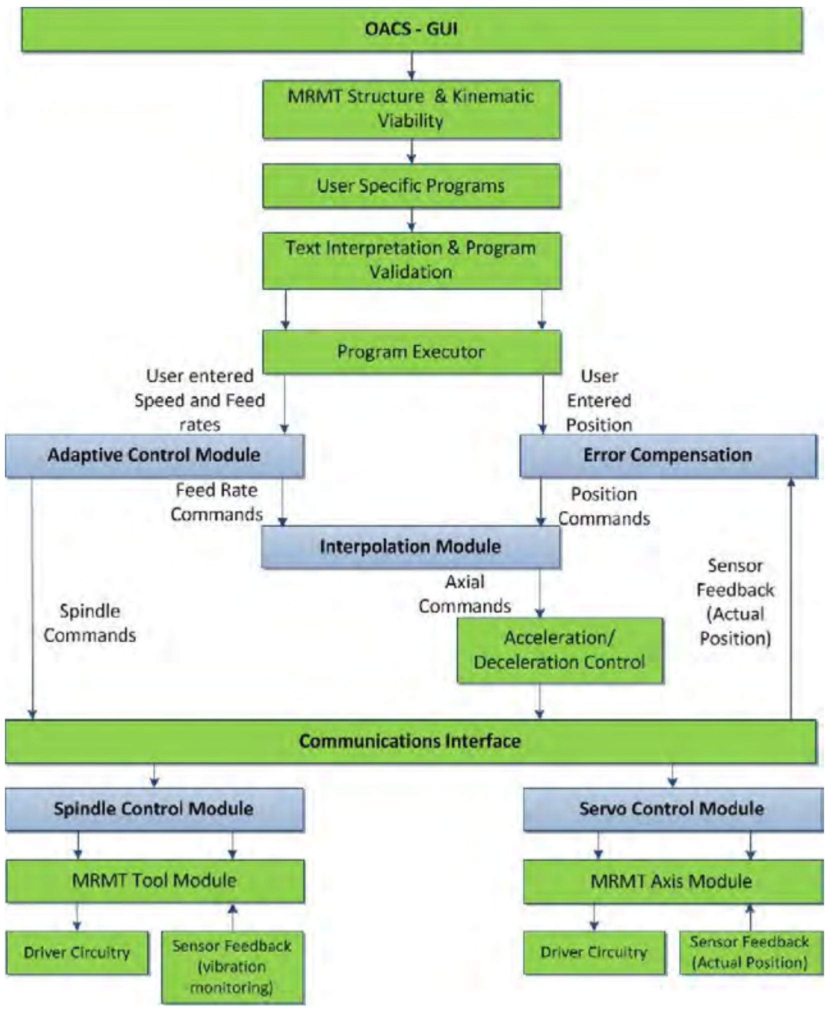


Figure 48: System Architecture Highlighting Control Functionality

Such a closed loop control system instructs the servo modules to carry out an instruction. However, often due to physical and mechanical limitations, the instruction may not be completed in the expected time. As a result, the controller uses the feedback data to compensate for the difference between desired position and actual position, and this is known as error compensation. The performance of the control module is therefore dependant on the following: mechanical and physical modules or axes, the response times of circuitry and the control algorithms. The feedback from the sensors is then fed back to the controller, and the servo motors on the MRMT are controlled to minimise the position error.

The closed loop feedback control system has three embedded control loops that are illustrated in Figure 49:

- The inner current loop;
- The middle speed control loop;
- The outer position control loop.

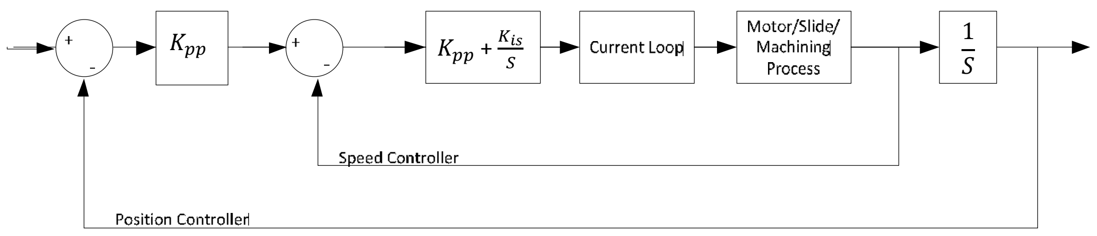


Figure 49: Closed Loop Feedback Control System

The cascade control loops can easily be tuned to improve system performance. However to ensure overall controller stability, it is necessary to ensure that the innermost loop is stable and the dependency between the outer loops and the inner loops is minimised. Furthermore the current loop, being the inner most control loop, needs to have a faster control loop response time than the outer loops [28]. This response times is configured via the controller tuning parameters. In contrast, the outer control loop, being the position controller, has the slowest response time.

The speed and current control is performed by the servo driver circuitry, and the position control loop is the control carried out by the control system. Since the control system controls the position control loop with the slowest response time, the machine movement and operation is highly dependent on the performance and response characteristic of the position control loop. Unfortunately, the slow response of the position control loop hinders performance and machining accuracy in systems of more than two axes [28].

7.2.1 PID Controller

The controller is named after the three control actions namely: the Proportional, the Integral and Derivative control actions. The main aim of the PID controller is to minimise the error between the desired position and actual machine position. The performance of the PID controller is highly dependent on the controller tuning. A further discussion on controller tuning is covered later on in the research.

The MRMT, due to its multiple axes, is actually a multiple input and output system. However, each axis can be controlled by an independent PID controller, as each axis is controlled separately based on its specific interpolated data received from the interpolator. Therefore, each axis is controlled by an independent PID controller with a single input and single output. Figure 50 illustrates the basic block diagram of a PID controller. The PID controller output (***u***) is first fed into the process, the process output (***y***) is then fed back to the block input where it is compared to the reference position (***r***), and the difference is the error (***e***) which is then fed back to the PID controller. The system then repeats the calculation.

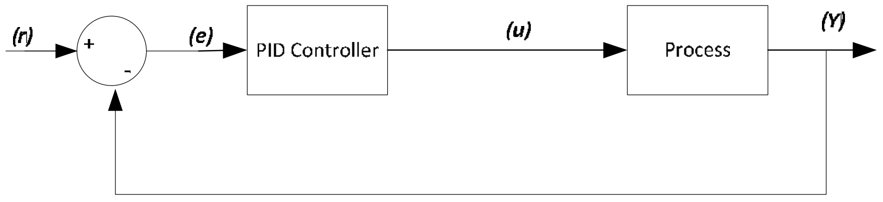


Figure 50: PID Block Diagram

The transfer function for the process, for a servo motor on an axis is given by equation 7.17 [58], where ***K*** is the electromotive constant, ***J*** is the moment of inertia of the mechanical system about the axis of the motor, ***b*** is a damping coefficient of the mechanical system on the axis, ***L*** is the motor inductance and ***R*** is the electrical resistance of the motor [59].

$$P(s) = \frac{K}{(Js+b)(Ls+R)+K^2} \tag{7.17}$$

The PID controller has three key parameters [60]. These parameters are listed with their corresponding functions:

- P – Proportional control:
  - The output is controlled depending on how far the actual position is from the desired position
  - The P control gain adds a sensitivity control due to the proportional gain, and makes the system more responsive to output error
  - The P control handles the immediate error
  - P control decreases the rise time and the steady-state error, but also contributes to overshoot
  - K_p – Proportional gain
- I – Integral Control:
  - The output is controlled depending on how long it takes to reach the desired position
  - As time to reach the desired position increases, the integral error builds up, and when the P control output drops, the integral control takes over and drive the machine to the desired position
  - The I control handles future errors
  - I control removes steady state error, decreases rise times, and causes an overshoot, but drives it back to the desired position
  - K_i – Integral gain
- D – Derivative control:
  - The output is controlled on the change in error
  - D control allows the controller to react to a sudden change in error and assists the system in maintaining a desired position
  - The D control handles errors based on the learning or what has been learnt from past errors
  - D control decreases overshoot and settling time
  - K_d – Derivative gain

The combined PID controller combines the three parameters, resulting in a controller that can accurately maintain a desired position [60]. Equation 7.18 [60] is the generic transfer function of a PID controller in the continuous domain.

$$G_c(s) = K_p + \frac{K_i}{s} + K_d S \tag{7.18}$$

Equation 7.19 shows the transfer function converted into the discrete time domain for digital control, where T represents the time period iterations.

$$G(z) = \frac{K_0 + K_1 z^{-1} + K_2 z^{-2}}{1 - z^{-1}} \qquad (7.19)$$

Where the constants $K_0 K\ _1$ and K_2 are:

$$K_0 = K_p + K_i T + \frac{K_d}{T} \qquad (7.20)$$

$$K_1 = -K_p - \frac{2K_d}{T} \qquad (7.21)$$

$$K_2 = \frac{K_d}{T} \qquad (7.22)$$

From Figure 51, the output of the PID controller can be represented as the difference equation 7.23 using the PID output and error input:

$$u(n) = u(n-1) + K_p\,(e(n) - e(n-1)) + K_i T e(n) + \frac{K_d}{T}\,(e(n)\ \ 2e(n-1) + e(n-2)) \qquad (7.23)$$

From equation 7.23 it can be seen that the PID controller at the current iteration looks at: the controller's previous input data, the current error multiplied by a constant, the error from the previous iteration multiplied by a constant, and the error from iteration before that is also multiplied by a constant.

As previously mentioned, the actual position data, received from the feedback sensors, which are the optical encoders on the axes, is fed back to the controller. The controller then compares the actual position with the desired position. The result of this comparison, the error $e = DesiredPosition - ActualPosition$ is then fed into the controller with the assumptions $u(0) = 0$ and $e(0) = 0$ and the result of the controller output is calculated according to equation 7.24 [60]:

$$u(n) = K_p\, e(t) + K_i \int e(t)\, dt + K_d\, \frac{d}{dt} e(t) \qquad (7.24)$$

The PID equation 7.24 can be used in the PID controllers for each axis, since each axis is independently controlled. The key difference at each iteration for each controller, is the reference data received, the desired position from the interpolater and the feed rates from the adaptive control block. Similar to the calculation to derive equation 7.24, the discrete time implementation for a PI controller can be shown as follows:

$$u(n) = K_p\, e(t) + K_i \int e(t)\, dt \qquad (7.25)$$

Users have the choice of selecting a PI or PID controller, alternatively a user may customize the system to include an additional controller based on the users specific requirements. Sample code of the implemented PID control routine can be found in Appendix C.

The integral error is limited to: within a range from -200 to 200, to counter the effects of integral wind-up. Integral wind-up occurs when the set point limit is reached and the PID controller continues to integrate for several iterations. As a result, the controller continues to integrate, increasing the integral error, and expecting to reverse the controller through the integral, however due to the wind up, the controller cannot respond. The problem occurs when the set point drops into a range when the controller should be able to respond, but since the integral error is above the limit, the controller cannot respond, and a lag is introduced until the controller has run for several iterations, where it has decreased the integral error to below the limits. To counter this lag and integral wind-up effect, the controller limits the integral error to a range of between -200 and 200 ensuring that the PID controller can respond immediately when the set point drops into a range where the controller can affect performance. Figure 51 [61] illustrates the windup phenomenon and the effects of lag.

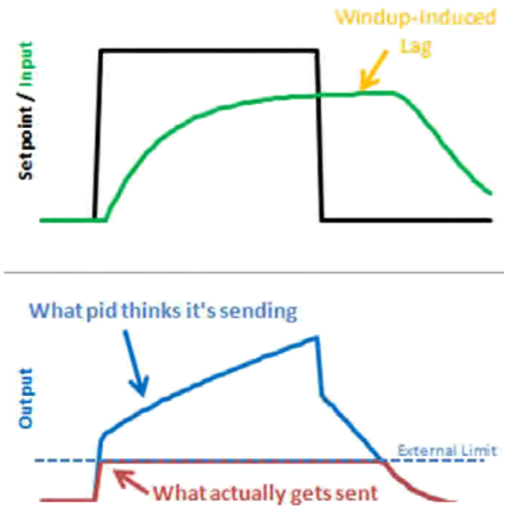


Figure 51: Controller Response

Similarly, the output of the PID controller is also limited to a range of 0 to 255. The output of the PID controller is used to set the duty cycle of the PWM signals required for the motor driver circuitry. The PWM duty cycles have a physical limitation from the microcontroller's specification, and therefore the PID controller has to act within that range. If the output is not limited to the limits of the PWM duty cycles then a similar problem, as described for integral windup occurs, resulting in lags introduced when the axis is nearing its set point. This lag makes the controller unstable, and overshoot the desired set point.

Additionally, the wiper motors on start-up had a stall current that had to be overcome. Therefore on the first iteration of the PID controller, the output is set to a maximum value and thereafter the output is manipulated as per the PID calculation. This was added in to ensure that the PID controller could operate from the very first iteration thereafter the speed controller would pull the speed back to the necessary point.

From the above, it can be seen that it is easy to implement a PID controller, but the performance of the PID controller is crucially dependant on the tuning and setting of the individual gains. Furthermore, users would have the ability to plot the response of their controllers in the program. They could then analyse the response and tune the gains appropriately. The easiest method to tune the gains is by trial and error, which includes manipulating the gains depending on the generic set of rules. Table 10 summarises the trial and error tuning guidelines of how changing each gain: ***Kp***, ***Ki*** and ***Kd*** affects performance.

Table 10: Trial and Error Tuning Guidelines

Response	Rise Time	Overshoot	Settling Time	S-S Error
Kp	Decrease	Increase	--	Decrease
Ki	Decrease	Increase	Increase	Eliminate
Kd	--	Decrease	Decrease	--

Table 10 can be used for the trial and error method to tune the PID controllers. However despite its simplicity the trial and error method is by no means the most accurate and efficient method of tuning control parameters. A more effective method used to tune controllers is the Ziegler-Nichols tuning method. It is however more complex and requires calculations.

The Ziegler-Nichols method was developed through experimentation by Ziegler and Nichols. They proposed a set of rules to determine the tuning parameters (***Kp***, ***Ki*** and ***Kd***) using the transient step response of the controller [62], also known as the reaction curve. The first Ziegler-Nichols method is for controllers with no integrators or complex conjugate poles, whose step response or reaction curve is S-Shaped, with no overshoot. Figure 53 [62] shows a reaction curve for a step response of a controller as well as the critical points required for Ziegler-Nichols tuning.

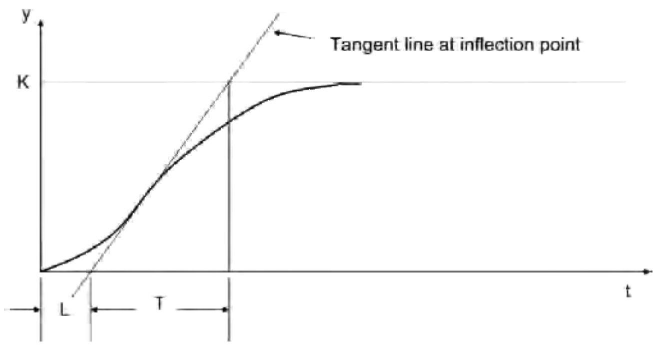


Figure 52: Ziegler Nichols Tuning

Using the Ziegler-Nichols method and the reaction curve, if a tangent line is drawn at the inflection point of the S-Shaped curve, two key constants are generated, namely the delay time ***L***, and the time constant ***T***. These two constants are determined by locating the intersections of the tangent line with the X-Axis, as well as the time and the steady state of the response curve [62]. Once these two constants have been calculated, Ziegler-Nichols proposes setting new values for ***Kp, Ki*** and ***Kd*** based on the formulas in the following table:

Table 11: Ziegler Nichols Tuning Rules

Controller	*Kp*	*Ki*	*Kd*
P	T/L	0	0
PI	$0.9T/L$	$0.27\,T/L^2$	0
PID	$1.2T/L$	$0.6\,T/L^2$	$0.6T$

7.3. Program Interpretation and Validation

The GUI presented to the user, has the option for the user to enter a user specific program for the current MRMT configuration. Full details on the GUI can be found in Section 8.3. The user entered program is interpreted and then validated against a selected set of NC code words. The validation ensures that the user entered program is executable on the current MRMT.

Table 12 covers the NC and reduced instruction set that the text interpretation and user program validation algorithms process:

Table 12: Reduced NC command Instruction Set

Function	Address	Example	Units
Program Number	P	P00001	--
Block Number	N	N01	--
Preparatory Function	G	G01	--
Coordinate (Translational Axis)	X, Z	X100, Z50	Mm
Coordinate (Rotary Axis)	A	A25	Degrees
Feed rate	F	F100	mm/rev – mm/min
Spindle Speed	S	S1000	Rpm

The following code is an example of a part program (P001) consisting of 4 NC blocks. Each block consists of several words, and each word consists of an address and a number. A part program such as this example can be programmed into the GUI and is interpreted and validated.

```
P001;
N10     G01     X0      Y0;
N20     G01     X100    A25     F100    S1000;
N30     M00;
N40     M02;
```

The part program and the NC words can be described as follows:

- N10 and N20 are the current code block numbers
- G01 is a preparatory function, in this case a movement command, which commands the relative movement between the tool and workspace

Thereafter, the individual axis can be instructed to move to a desired location, as is demonstrated in this example.

- In the code block N10, the X and Y axes are commanded to move to the origin or zero point at the default feed rates
- In the code block N20, the X axis is requested to move to 100 mm towards the workspace from the origin or zero point and the rotary axis A, 25 mm clockwise from its origin or zero point

Finally, the F-code and S-code is entered, setting the feed rates for the axes and the spindle speed for the tool head module. As per the example for code block N20:

- The feed rate is set at 100 mm/rev;
- The spindle speed is set to 1000 rpm;
- The “;” is the end of block character.

Code blocks N30 and N40 demonstrate the movement start and stop commands. The flow chart in Figure 53 illustrates the sequence of events followed for text interpretation and user program validation. Once the user has entered in the program, and the program is saved, the text interpretation and program validation routine is called. The routine opens a .dat file and updates it as the interpretation and validation is processed. Each code block is read sequentially and each word is verified. The words are checked to see if they can be classified as:

- M or G code words
  - Preparatory functions for modes or machine commands
- Coordinate commands
  - Verification against the limits of the respective axis, depending on the current axis position to ensure that the MRMT can carry out this command
- Feed rates
  - Verification to ensure that the feed rate is less than the lowest feed rate of all the axes on the MRMT.
- Spindle speeds
  - Verification to ensure that the speed commanded is within the range of operation
- End Of Block (EOB commands)
  - Signals end of the current code block

As each line of code is read, interpreted and verified, it is saved in the .dat file for machine operation. During the verification, if NC code block fails to be interpreted or verified, the user is presented with an error message, signalling that the code block is invalid. The text interpretation and validation routine is then terminated and the .dat file is deleted. Sample code for the text interpretation routine is attached in Appendix D.

The text interpretation is implemented for the reduced instruction set. Depending on the MRMT requirements, a more complex text interpretation can be implemented on the OACS. If required, the OACS can be modified by simplify editing the text interpretation routine only, ensuring that the pointers to the input data and output data in the routines follows the conventions of the original text interpretation routine. Changing this routine does not affect the performance and operation of the rest of the OACS, thus making the OACS reconfigurable and customizable as per the user requirements.

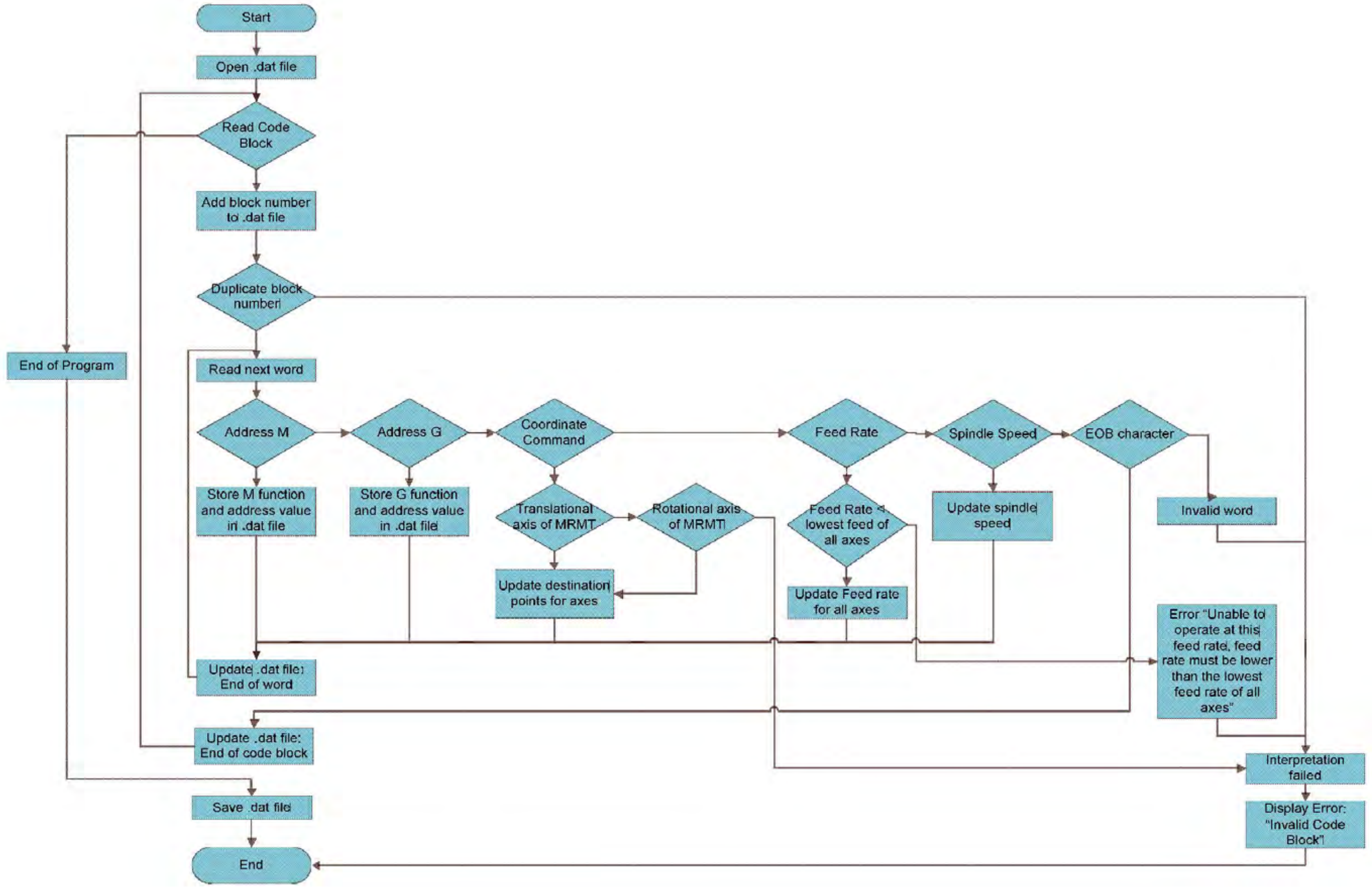

Start
Open .dat file
Read Code Block
Add block number to .dat file
Duplicate block number
End of Program
Read next word
Address M
Address G
Coordinate Command
Feed Rate
Spindle Speed
EOB character
Store M function and address value in .dat file
Store G function and address value in .dat file
Translational axis of MRMT
Rotational axis of MRMT
Feed Rate < lowest feed of all axes
Update spindle speed
Invalid word
Update destination points for axes
Update Feed rate for all axes
Error "Unable to operate at this feed rate, feed rate must be lower than the lowest feed rate of all axes"
Update .dat file: End of word
Update .dat file: End of code block
Interpretation failed
Save .dat file
Display Error: "Invalid Code Block"
End

Figure 53: Text Interpretation and Validation Flowchart

7.4. Interpolation

The interpolator determines the movement requirements of each axis depending on the: current position, the destination or target point and the maximum speeds and feed rates of each axis. The user entered program is read, and verified after which it is saved in a .dat file. The interpolator accesses the movement commands entered by the user, compares the movement requirement to the current MRMT axes positions and determines the optimum movement commands for the axes and the required feed rate for each axis.

7.4.1. Linear Interpolation

Linear interpolation is used to move the MRMT axes from a start position to a destination position using the shortest distance possible. The motion is a straight line from the start point to destination point. As mentioned previously interpolation is the process of calculation of intermediate coordinate points between start and end positions that follow the shortest path along the contour. The interpolation routing controls the feed rates of each axes and updates the expected position via updating coordinates.

Linear interpolation is computed for a single or two axial motions and each block of code that requires axial movement uses the interpolation routine. The current machine tool axial coordinates, the destination coordinates and the feed rates of each axis are required by the interpolator to compute the interpolation coordinates. The easiest of the three is the single axis linear interpolation as in single axis interpolation, the tool movement is always parallel to the axis [63].

Figure 54(a) illustrates the basic operation of an interpolator for point to point control method for a 2 axial movement requirement. The two axes linear interpolation is a very common movement requirement for CNC systems and is used to further explain the interpolation computations. The MRMT needs to move from P1 to P2. In order for a successful accurate movement, the interpolation routine requires the following [28]:

- Interpolation data needs to match the actual part shape;
- Due to mechanical structure and servo motor speed limitations on the MRMT, the interpolator needs to calculate velocity and feed rates that all axes can operate at.

Figure 54(b) shows an example of the interpolated results based on the movement requirements. The total movement required by each axis is divided up into equal segments, and based on the lower feed rate of the two axes, a constant time interval and the velocity of each axis calculated. The result is a computation based on the shortest path between the two points. The illustrated result of this calculation ensures that the two axes reach the destination point after the same time interval, despite different movement distances for each axis via manipulation of the feed rates of each axis.

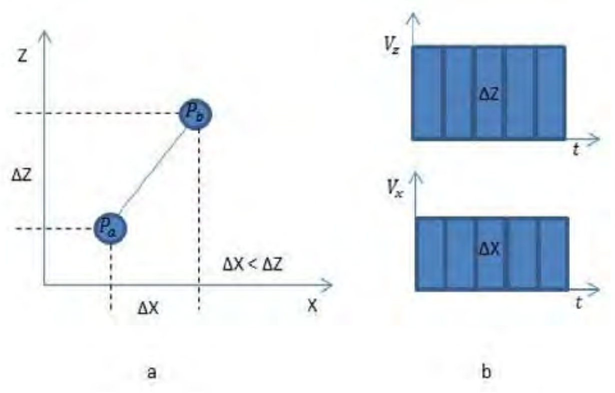


Figure 54: (a) 2 Axis Input Parameters (b) Interpolated results

To ensure that all the axes reach the destination point simultaneously and that the MRMT reaches the destination by travelling along the shortest possible path, the interpolator calculates the feed rate of each axis individually. The interpolator either "speeds up" or "holds back" an axis depending on direction of motion to ensure that all axes reach the destination simultaneously.

The interpolation routine has been developed on the reference word interpolator [63]. Figure 55 [63] shows the critical information required for a linear interpolation calculation. The actual tool motion follows the path ***L***, the length of ***L*** can be calculated by Pythagoras theorem for right angled triangles. The following is a summary for the derivation of equation 7.29 and equation 7.31 [28].

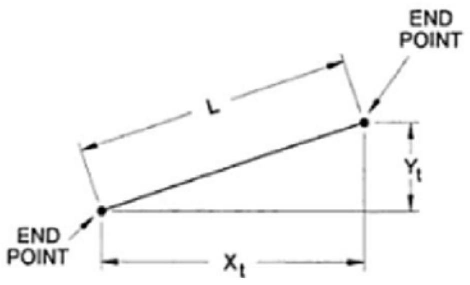


Figure 55: 2 Axis Linear Interpolation Example

The interpolation routine calculates the required feed rate for each axis by the following equation 7.26, where F is the lowest feed rate of all the axes.

$$\boldsymbol{F_X = \frac{X_t}{L*F} \quad or \quad F_Y = \frac{Y_t}{L*F} \quad or \quad F_Z = \frac{Z_t}{L*F}} \tag{7.26}$$

Central to the interpolation routine calculation is division of the axis displacement by the interpolation time, $\boldsymbol{t_{ipo}}$. The total displacement $\boldsymbol{X_t}$ or $\boldsymbol{Y_t}$ is then divided by the interpolation time $\boldsymbol{t_{ipo}}$ to reveal the Figure 56 [28].

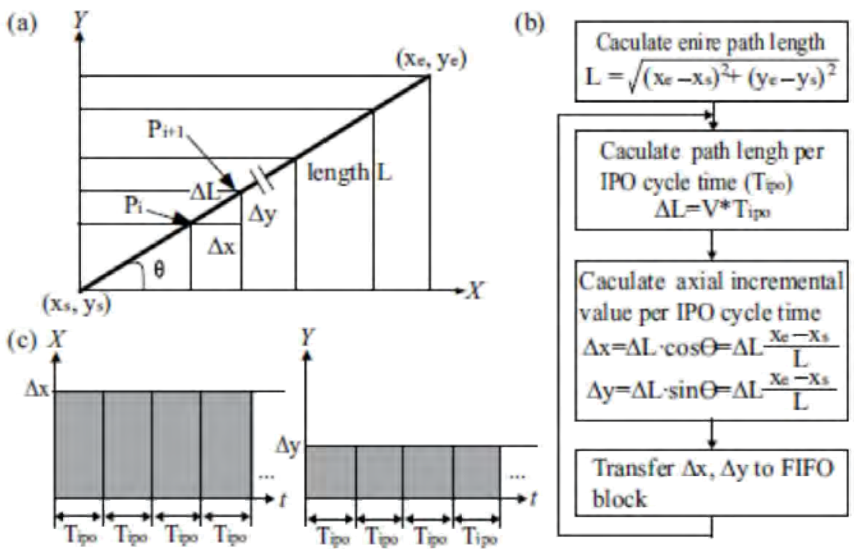

(a)
Y
(xe, ye)
Pi+1
ΔL
Pi
Δy
Δx
length L
θ
X
(xs, ys)
(b)
Caculate enire path length
L = √((xe − xs)² + (ye − ys)²)
Caculate path lengh per IPO cycle time (Tipo)
ΔL=V*Tipo
Caculate axial incremental value per IPO cycle time
Δx=ΔL·cosΘ=ΔL(Xe−Xs)/L
Δy=ΔL·sinΘ=ΔL(Xe−Xs)/L
Transfer Δx, Δy to FIFO block
(c) X
Δx
Y
Δy
t
Tipo Tipo Tipo Tipo
Tipo Tipo Tipo Tipo

Figure 56: Reference Word Algorithm

The area under the curves then equal velocity of each axis and can be converted to the feed rates:

$$\Delta x = \frac{f\,T_{ipo}}{60} \tag{7.27}$$

The position of each axis at a given time can then be determined by:

$$x_{i+1} = x_i + \Delta x \tag{7.28}$$

This residual length can be calculated as follows:

$$x_{residual} = X - v_x \tag{7.29}$$

The total number of iterations required by the interpolator is calculated by:

$$N = \frac{L}{\Delta L} \tag{7.30}$$

N is always rounded up to ensure that the residual displacement required is calculated and the respective commands sent out by the interpolator. When the interpolation routine runs for the N^{th} time, the new feed rate required for the final iteration can be calculated as follows:

$$f_x = \frac{60\,x_{residual}}{T_{ipo}} \tag{7.31}$$

The interpolation routine runs and calculates the movement requirements for each axis based on these calculations and the results are stored in a First In First Out (FIFO) buffer. The preparatory command G01 is used for linear interpolation. Sample code for the linear interpolation routine is attached in Appendix E.

7.4.2. Circular Interpolation

Circular interpolation is a control method for CNC machines to controlling a machine tool along an arc or a complete circle depending on machining requirements. A circle is defined by its centre point and its radius. The preparatory commands G02 and G03 are used for circular interpolation, with G02 for Clockwise motion and G03 for counter clockwise motion.

The code block specifies the preparatory command, the plane of interpolation and the start and end points which also defines the arc radius. The circular interpolator approximates the circular path or arc by small straight line segments. The arc movement accuracy is dependent on the number of line segments, although a trade-off between computation and accuracy is required as more interpolation iterations are required for more line segments.

The reference word interpolator for circular interpolation was implemented and the velocity of the MRMT during circular interpolation can be calculated as per the following derivation. The following is a summary for the derivation of equation 7.44 and equation 7.45. The axes velocity is calculated by the interpolation routine and used as the reference input for the position controllers.

$$\boldsymbol{V_x(t) = V\, sin\theta(t)} \tag{7.32}$$

$$\boldsymbol{V_y(t) = V\, cos\theta(t)} \tag{7.33}$$

$$\text{where } \boldsymbol{\theta(t)} = \boldsymbol{\frac{Vt}{R}} \tag{7.34}$$

Figure 57 [28] illustrates the data for two successive iterations for the circular interpolation routine. As mentioned, there exists a trade-off between accuracy, computational power and the angle $\boldsymbol{\alpha}$ is the determining factor for the number of iterations. The following equations can be extracted from Figure 57, using the theorem of Pythagoras for right angled triangles.

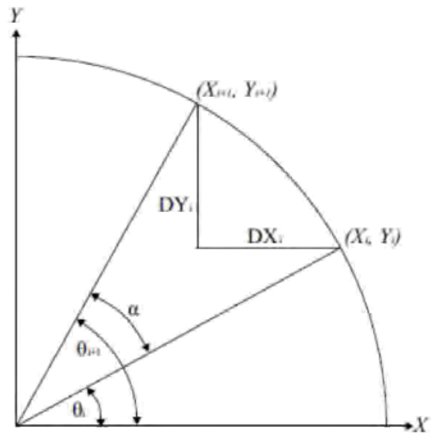


Figure 57: Reference Word Algorithm for Circular Interpolation

$$\boldsymbol{\cos\theta_i = \frac{X_i}{R}} \tag{7.35}$$

$$\boldsymbol{\sin\theta_i = \frac{Y_i}{R}} \tag{7.36}$$

$$\boldsymbol{\theta_{i+1} = \theta_i + \alpha} \tag{7.37}$$

Similarly, manipulating equation 7.35 and equation 7.36 we get equation 7.38 and equation 7.39:

$$\boldsymbol{X(i+1) = \mathrm{R(i)} \cos\theta_{i+1}} \tag{7.38}$$

$$\boldsymbol{Y(i+1) = R\,(\mathrm{i})\; sin\theta_{i+1}} \tag{7.39}$$

Using the angle sum trigonometric identities and combining the results under the assumption ($\boldsymbol{A = \cos\ \alpha}$ and $\boldsymbol{B = sin\ \alpha}$):

$$\boldsymbol{X(i+1) = AX(i) - BY(i)} \tag{7.40}$$

$$\boldsymbol{Y(i+1) = AY(i) + BX(i)} \tag{7.41}$$

Now using equation 7.40 and equation 7.41, and from Figure 58, the next interpolation coordinates can be calculated as follows:

$$\boldsymbol{DX(i) = X(i+1) - X(i) = (A-1)X(i) - BY(i)} \tag{7.42}$$

$$\boldsymbol{DY(i) = Y(i+1) - Y(i) = (A-1)Y(i) + BX(i)} \tag{7.43}$$

Finally, using Pythagoras theorem, the axial increments can be computed and the interpolation routine can calculate the individual axes velocities as follows:

$$\boldsymbol{V_x(i) = \frac{V\,DX(i)}{DS(i)}} \tag{7.44}$$

$$\boldsymbol{V_y(i) = \frac{V\,DY(i)}{DS(i)}} \tag{7.45}$$

Where

$$\boldsymbol{DS(i) = \sqrt{DX(i)^2 + DY(i)^2}} \tag{7.46}$$

To solve for $\boldsymbol{\alpha}$, **A** and **B**, a number of sampled data interpolation methods can be chosen with trade-offs between iterations and the accuracy. Suh et al compares six different methods and based on this comparison [28], the Improved Tustin method was chosen due to the lower number if iterations and accuracy.

The improved Tustin method makes the radial error ***ER*** and chord height shown in Figure 58 [28] equal to 1. In doing so, the angle $\boldsymbol{\alpha}$ can be calculated as follows:

$$\boldsymbol{\cos\frac{\alpha}{2} = \frac{R-1}{R+1}} \tag{7.47}$$

Solving for α:

$$\boldsymbol{\alpha \cong \frac{4}{\sqrt{R}}} \tag{7.48}$$

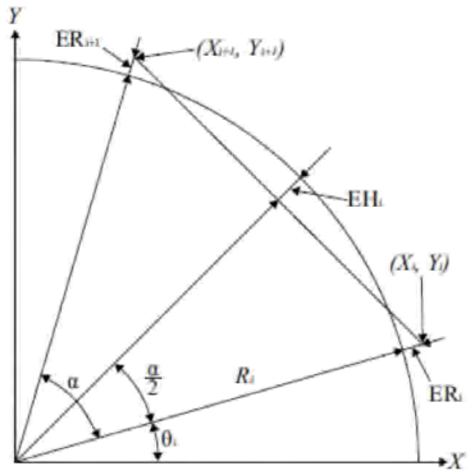


Figure 58: Radial and Cord Height Errors for Circular Interpolation

And the number of iterations:

$$\boldsymbol{N = \frac{\pi}{8}\sqrt{R}} \tag{7.49}$$

Sample code for the circular interpolation routine is attached in Appendix F.

7.5. Acceleration and Deceleration Control

If the current OACS were implemented as is, the axis would approach its destination and abruptly stop. In order to prevent this mechanical shock and to ensure smooth axes control, an acceleration and deceleration routine and control algorithm has been implemented.

The interpolation routine calculates the axis displacements after which the acceleration and deceleration routine performs the acceleration and deceleration on the axial commands.

Digital filter theory is used as the basis for the acceleration and deceleration control algorithm, where if an input signal $\boldsymbol{x}[\boldsymbol{n}]$ is the input to a filter with impulse response $\boldsymbol{h}[\boldsymbol{n}]$ then the filter has an output $\boldsymbol{y}[\boldsymbol{n}]$ which is the convolution of $\boldsymbol{h}[\boldsymbol{n}]$ and $\boldsymbol{x}[\boldsymbol{n}]$ [28]. For a discrete system, the general convolution is shown in equation 7.50, equation 7.51 and equation 7.52 [28].

$$\boldsymbol{f}[\boldsymbol{n}] = \boldsymbol{f}_1[\boldsymbol{n}] * \boldsymbol{f}_2[\boldsymbol{n}] \tag{7.50}$$

$$\boldsymbol{f}[\boldsymbol{n}] = \boldsymbol{f}_1[\mathbf{0}]\boldsymbol{f}_2[\boldsymbol{n}] + \cdots + \boldsymbol{f}_1[\boldsymbol{k}]\boldsymbol{f}_2[\boldsymbol{n}-\boldsymbol{k}] + \cdots + \boldsymbol{f}_1[\boldsymbol{n}]\boldsymbol{f}_2[\mathbf{0}] \tag{7.51}$$

$$\boldsymbol{f}[\boldsymbol{n}] = \textstyle\sum_{\boldsymbol{k}=\mathbf{1}}^{\boldsymbol{n}} \boldsymbol{f}_1[\boldsymbol{k}] * \boldsymbol{f}_2[\boldsymbol{n}-\boldsymbol{k}] \tag{7.52}$$

If $\boldsymbol{x}[\boldsymbol{n}]$ denotes the interpolation output and $\boldsymbol{h}[\boldsymbol{n}]$ be the impulse response with a normalized unit summation, the convolution results is illustrated in Figure 59 [28] where for discrete time, $\boldsymbol{\tau}$ is the product of n and the sampling time **T**.

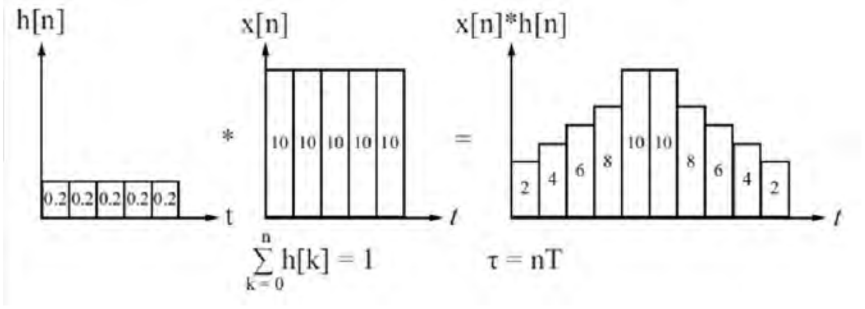


Figure 59: Convolution Example

For acceleration and deceleration control the calculated pulse train is dependent on the type of impulse response function used. The three common impulse responses are the linear type, exponential type and the S-shape type as show in Figure 60 [28].

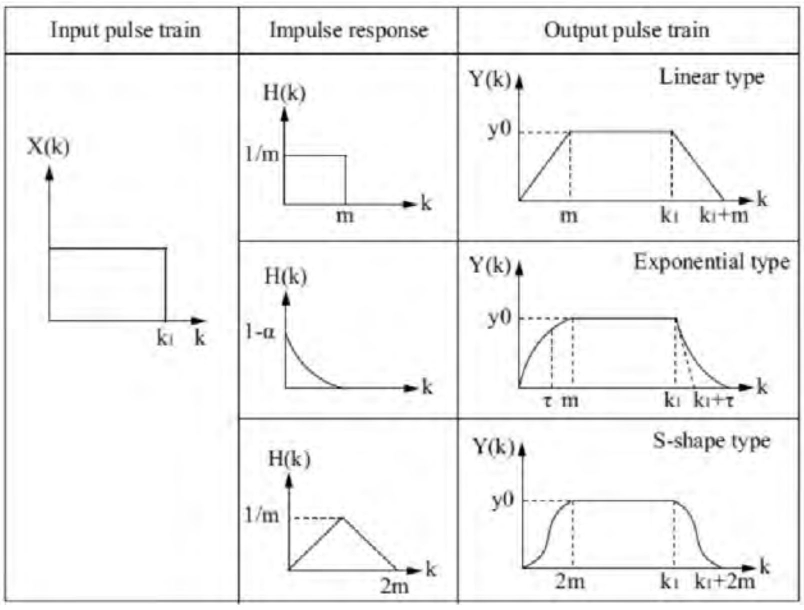


Figure 60: Input and Output Pulse Train Profiles

The S-shape method was chosen and implemented for the acceleration and deceleration control routine as the S-Shape translates to a smoother acceleration and deceleration control of the axis. In addition as is seen in an example, the S-Shape method can result in a linear type response if desired by manipulation of the routine constants. The S-shape method can best be described in reference to Figure 61, which shows a hardware implementation for the algorithm but can easily be converted to a software calculation.

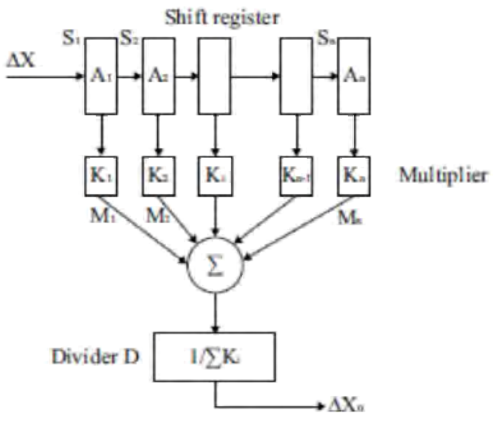


Figure 61: Illustration for S-Shape Acceleration/Deceleration Control [28]

From Figure 61 the output $\Delta \mathbf{X_0}$ is computed as equation 7.53 where $\mathbf{K_i}$ is the multiplier values, and $\boldsymbol{A_i}$ the input data shifter at the $\boldsymbol{i^{th}}$ iteration.

$$\Delta \boldsymbol{X_0} = \frac{\sum_{i=1}^{n} A_i K_i}{\sum_{i=1}^{n} K_i} \tag{7.53}$$

$\Delta \boldsymbol{X_0}$ defines the acceleration and deceleration output pulses which are then stored in a first in first out buffer file for transmission to the respective distributed control modules. The code for the acceleration and deceleration routine is attached in Appendix G.

7.6. *Encoder Position Algorithms*

The index pulse is generated once per rotation of the code wheel, and the channel A and channel B pulses are generated in a quadrature format as shown in Figure 62. The 4 pulses generated by Channels A and B are each generated 256 times in every rotation adding up to a count of 1024 per revolution.

The direction of rotation can be determined by analysis of the quadrature waveforms, if the code wheel viewed from the encoder end of the motor and the code wheel is rotating in a counter clockwise direction then channel A leads channel B. Similarly if the code wheel is rotating in a clockwise direction then channel B leads channel A [51].

The output signals from the three channels are connected to the external interrupt pins on the distributed modules and when the interrupts trigger a counter is incremented or decremented depending on the result of this check.

The incrementing and decrementing of the counter imply a movement of the axis which has a zero set point located at the midpoint of the axis. From Figure 62 [51] it can be seen that when Channel A triggers, the direction of rotation is actually determined by the value of Channel B, similarly when Channel B triggers, the direction of rotation is determined by the value of Channel A. Table 13, summarises the decoding of the quadrature waveforms from the encoders [51].

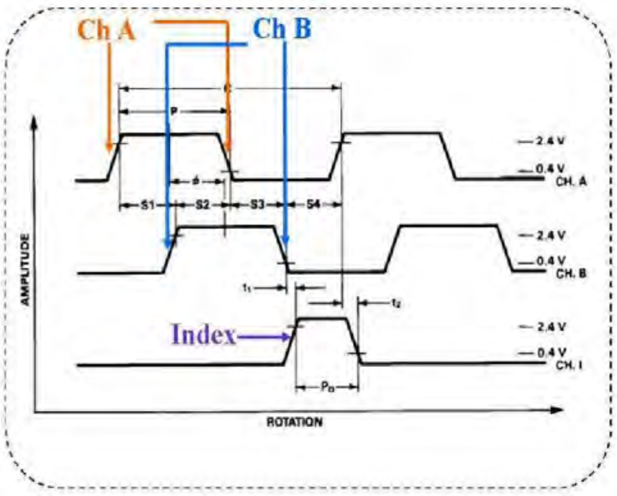


Figure 62: Output Signals from the Encoder

Table 13: Logic for Decoding Quadrature Encoder Waveforms

Channel	Interrupt 1	Rotation	Interrupt 2	Rotation
A	B = Low	Clockwise	B = High	Counter Clockwise
B	A= Low	Counter Clockwise	A = High	Clockwise

The code for interrupt service routine on the Arduino UNO for Channel A is shown below.

```
void EncoderAISR()
{
  // check channel B to see which way encoder is turning
//if Channel B is LOW then Clockwise rotation
    if (digitalRead(EncoderB) == LOW)
```

```
      {
        EncoderPos = EncoderPos + 1;         // CW
      }
      Else
     //channel B is High, CounterClockWise Rotation
      {
        EncoderPos = EncoderPos - 1;         // CCW
      }
  }
```

7.6. Chapter Summary

Chapter 7 has covered the various control and interpretation algorithms that the OACS uses from start-up, configuration to machining. The algorithms covered include the kinematic viability routine, control routines, text interpretation and user program validation, interpolation and acceleration and deceleration routines. Examples of the code for the routines are also presented to illustrate how the OACS actually implements the theory.

8. Open Architecture Control System Software

Chapter 8 covers a discussion on the GUI of the OACS. The various functions of the OACS is discussed and the value add of each function highlighted. Furthermore a flowchart of sequence of events that the user is expected to follow is presented which closely relates to the function of the OACS. Finally the distributed module software is also covered, with a flowchart to illustrate what the distributed modules are required to manage.

8.1 Overview

For the benefits of an OA system to be realized, a high correlation between RMS systems and the control system needs to exist in both hardware and software. A MRMT, being a derivative of the RMS paradigm, is designed and developed on the core principles of reconfigurability, modularity and others.

Similarly, through the: research, design and development of the open control system, the aim was to incorporate the following core principles in all aspects:

- Extendibility:
  - The ability of a number of modules to run on a system without conflict
- Interoperability:
  - The integration of modules such that the modules function in a consistent way, and communicate via pre-defined protocols of data exchange
- Scalability:
  - The ability to add, integrate, remove or swap in new modules thereby scaling the system up or down to adapt the performance or functionality of the system
- User Customization

The following aspects were crucial to the design, development and realization of a control system with the aforementioned characteristics: The proposed and implemented solution of distributed modules in the field, derived classes, software modules and the one to one linking.

By creating a modular electronic hardware and software system with well-defined interfaces, the control system has the ability to be scalable in all aspects. Distributed modules can be added on to the communications network with ease. Furthermore by scanning the network, the host PC discovers this new module and create an associated class for it. Alternatively modules can also be removed or swapped, thereby satisfying the scalability requirements of the control system.

The implementation of the modular software and modular hardware system has also ensured that the control system is indeed extendable. The design philosophy is based on a one to one link between software and hardware. This approach ensures that a number of modules are capable of running on the system without conflict.

The control system aims to address the customization requirements of open control systems by including three methods to customize programs and control algorithms. As discussed previously, users have the option to: program, compile and run custom applications for their MRMT. This allows for the evaluation of the performance of the controllers and if need be the controllers can be tuned by the user. These three methods provide flexibility, reconfiguration and customization to the end user, which are critical evaluation aspects of open control systems.

8.2 Development Environment

At the head of the system is the host PC. The host PC is responsible for interfacing all the distributed modules, which in turn interface the mechanical axis on the MRMT, as depicted in Figure 63.

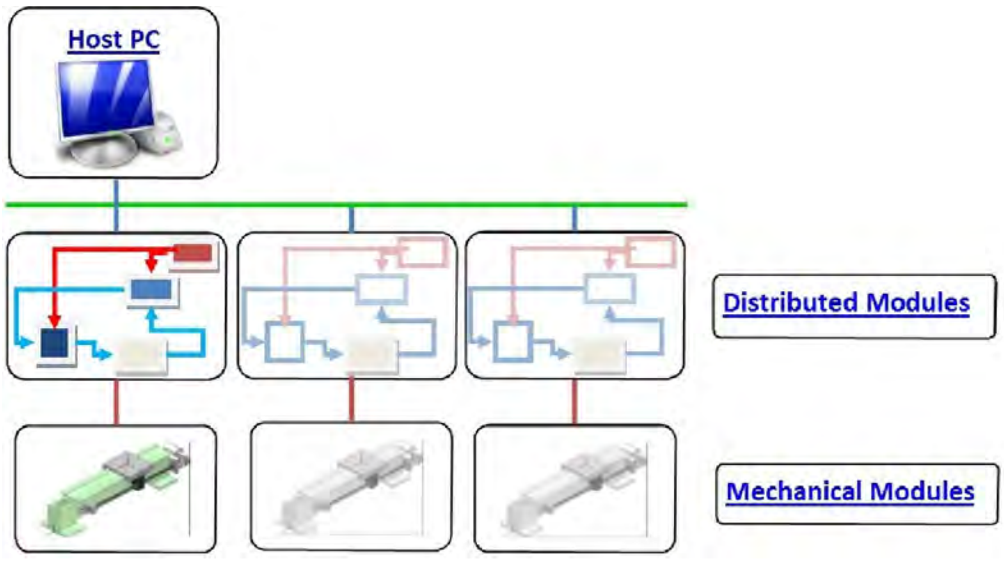


Figure 63: System Architecture

The host PC software was developed using Microsoft Visual C#, which is based on Microsoft's .Net Framework. As highlighted in Chapter 6, the choice to have cross platform uniformity and standardisation, as well as ease of understanding, the use of a common language and framework was preferable. Furthermore, the CAN bus communication protocols have standard libraries for C# application, which allows the user to set up and utilise the full features of the CAN bus hardware.

In addition the applications developed with Microsoft Visual C# have well defined external interfaces and support for external inputs and interfacing with other software. Although this does not form part of the scope of this research, the use of Visual C# can assist further research and development for the integration of external interfaces. One of the aims, although not part of this research, is to enable the OACS to have open external interfaces for users to load user specific control algorithms and programs between different software platforms. The well-defined visual C# interfaces assists in the implementation of this feature of OACS.

8.3 User Interface

The user interface, also known as the Human Machine Interface (HMI), or the GUI, developed in Microsoft Visual C#, uses windows forms to present the user with an interface that is user-friendly.

When the OACS is initially run, a GUI is presented to the user. The tabbed GUI is presented with different configuration and setup tabs for the user sequentially step through to configure the MRMT. The start-up GUI that the user initially sees is shown in Figure 64.

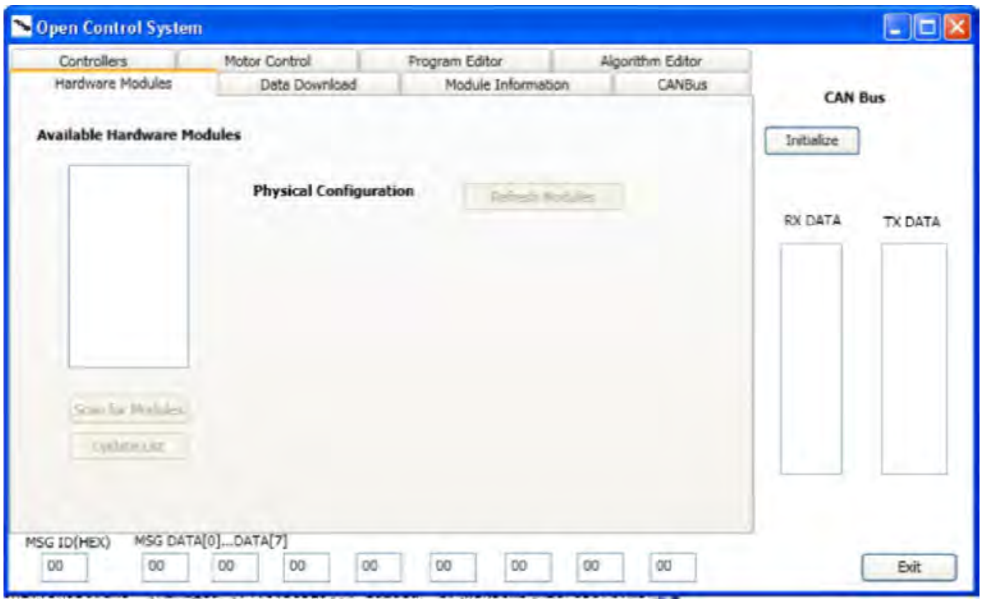


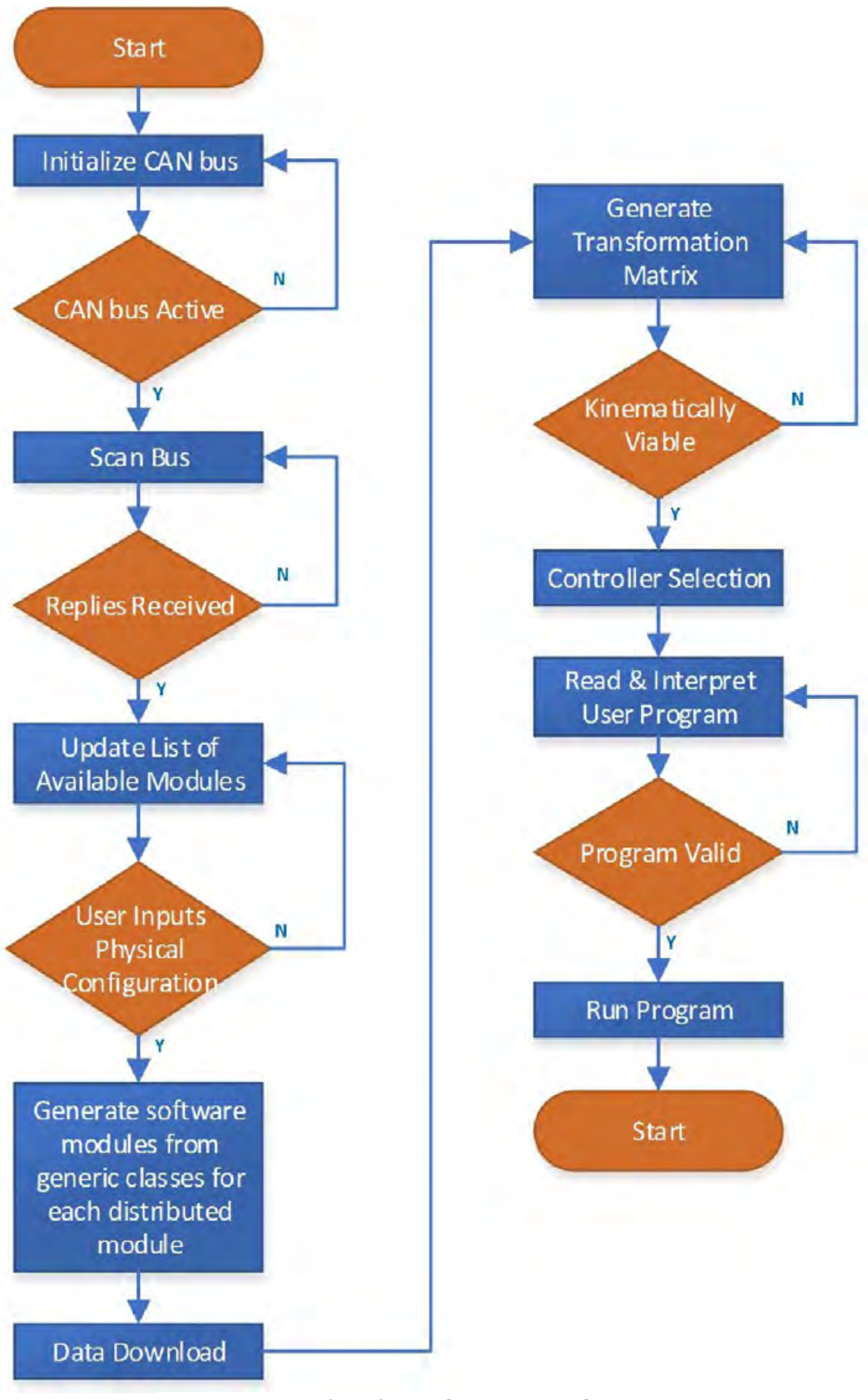
Figure 64: Graphical User Interface

Due to the non-determined and dynamic structure of the MRMT, the user must follow the step by step setup and configuration process to create the software for the structure of the MRMT as well as the control requirements for the host PC.

The GUI shows the user several tabs that are followed sequentially during set up. The following tabs are available for the user:

- Hardware Modules:
    - After initialization of the CAN bus network, a bus scan is run to determine which modules are connected and active, after which the user is to enter the physical configuration of the MRMT to determine kinematic viability
- Data Download:
    - Host PC downloads critical information such as control requirements, limitations and transformation matrices, from each connected module
- Module Information:
    - Displays information downloaded from each module
- CAN bus:
    - To assist with debugging and diagnosis. The tab displays the raw CAN bus data received
- Controllers:
    - Controller selection tab
- Motor Control:
    - To assist with tuning and debugging. Individual control and movement of each connected module can be setup
- Program Editor:
    - Tab to allow the user to enter a custom program
- Algorithm Editor:
    - Interface to allow the user to customize and tune the control parameters on each distributed module

In order to better demonstrate the flow of operations that are run, the flow chart in Figure 65 illustrates the sequence of events on start-up. Sample code of the CAN bus class for the OACS is attached in Appendix H where initialisation, general call, and data download calls and others have been implemented.



Start
Initialize CAN bus
CAN bus Active
N
Y
Scan Bus
Replies Received
N
Y
Update List of Available Modules
User Inputs Physical Configuration
N
Y
Generate software modules from generic classes for each distributed module
Data Download
Generate Transformation Matrix
Kinematically Viable
N
Y
Controller Selection
Read & Interpret User Program
Program Valid
N
Y
Run Program
Start

Figure 65: Flowchart of Sequence of Events

8.4 Set-Up and Configuration

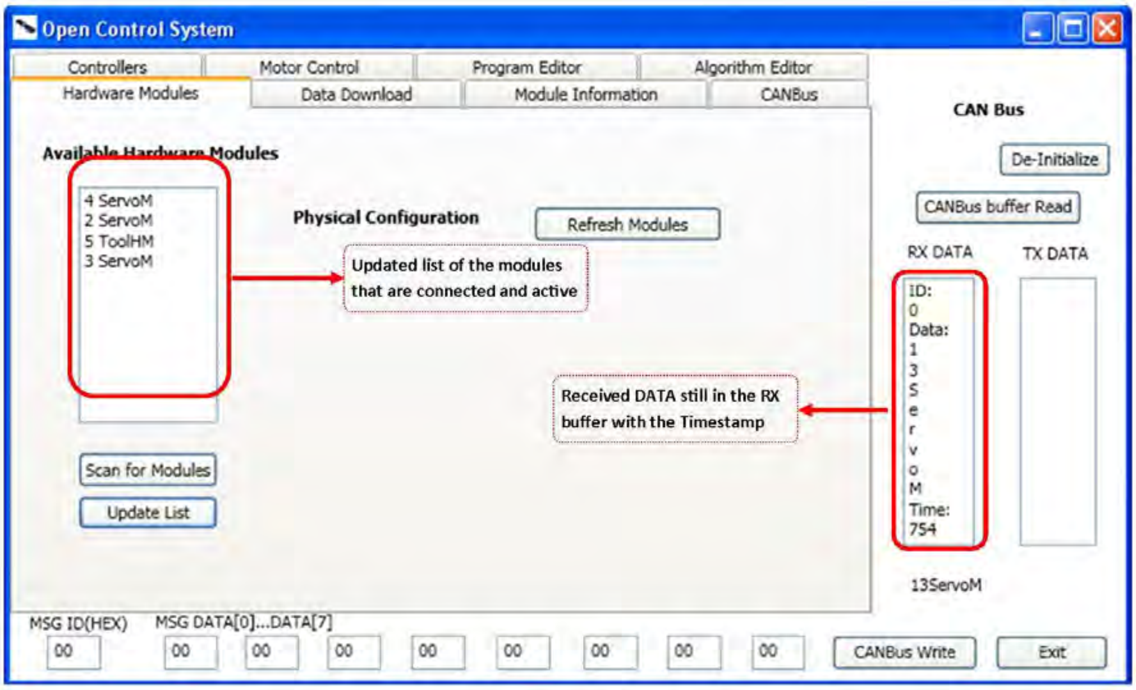
The system on start-up automatically determines which hardware modules are connected and active by conducting a scan on the bus. Each distributed module that is connected to the bus replies with its unique ID, thereby signalling its active status on the bus. Additionally, with active status and signal, a predefined code is also be transmitted. This code informs the host PC of the type of module it is, e.g. servo or spindle modules.

Each of the distributed modules have a dedicated software class object derived from a generic class created for it. In order for this newly generated class to inherit the correct functions and parameters from the generic class, the type of module needs to be known. Tables showing examples of the code words transmitted have been covered in Section 6.7.

The flowchart in Figure 65 is a representation of the sequence of events that the OACS follows during initialisation or reconfiguration process. The following sub-chapters cover each step in the initialisation or reconfiguration process in more detail with screenshots of the GUI indicating the expected data that the user sets up.

Figure 66 shows an example of the results received from a scan, where four modules are connected to the bus. The same four modules are connected to the system in all GUI screenshots in this chapter for consistency and to illustrate how the OACS handles the connected modules. The four connected modules are as follows:

- Three servo modules
    - Two translational axis
    - One rotary axis
- One tool module
    - Drill module



4 ServoM
2 ServoM
5 ToolHM
3 ServoM

ID:
0
Data:
1
3
S
e
r
v
o
M
Time:
754

13ServoM

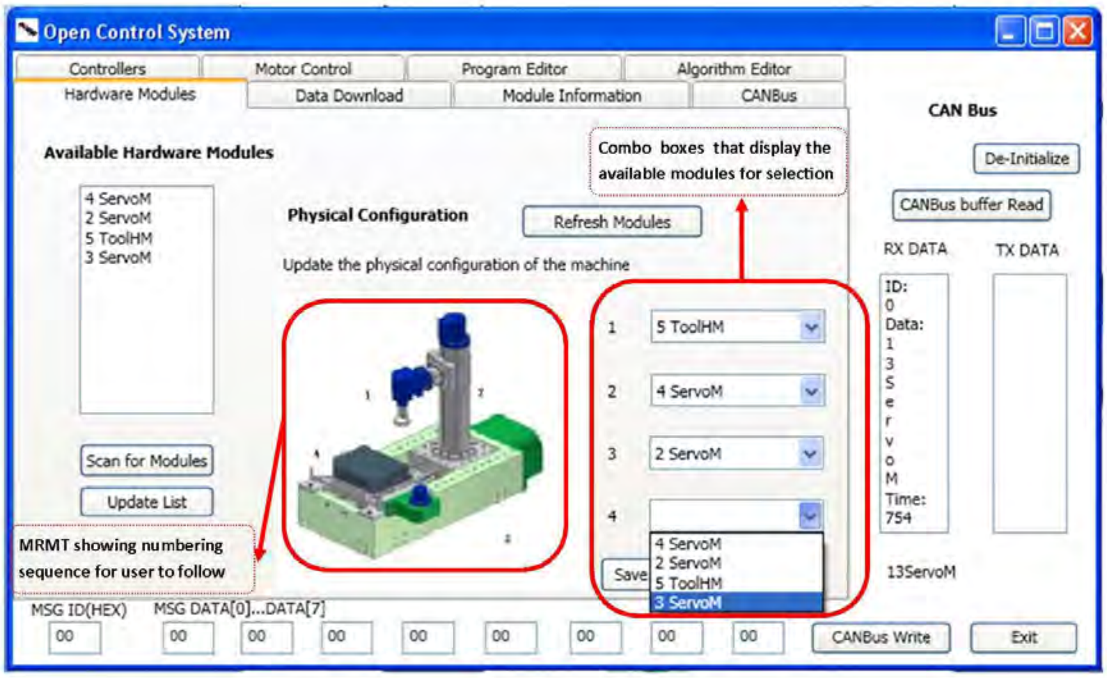
00 00 00 00 00 00 00 00 00

Figure 66: GUI Showing Results of a Bus Scan

The data that is received from the distributed modules is stored in temporary classes. This is because the host PC does not yet know the physical configuration of the MRMT. It is necessary to know the physical configuration of the MRMT, as a computation of the transformation matrix of the current machine structure is required for control purposes. Furthermore, the complete transformation matrix of the machine tool is used to determine if the structure of the machine tools is kinematically viable [29].

After scanning the bus, the host PC has complete list of all the modules that are active and connected. Thereafter, using this list, the host PC asks the user to input the details of the physical configuration of the hardware modules. The numbering sequence starts at tool head and work towards the product workspace as required, in order to determine the kinematic viability as depicted in Figure 67.

Sample code for a generic servo module class is attached in Appendix I where the methods and variables of a generic software class can be found. New classes are derived from this generic class at runtime depending on the user entered configuration and the available hardware modules. The temporary software classes contain similar methods and variables as the generic servo class shown in Appendix I.



Combo boxes that display the available modules for selection

MRMT showing numbering sequence for user to follow

Update the physical configuration of the machine

Available Hardware Modules: 4 ServoM, 2 ServoM, 5 ToolHM, 3 ServoM

1 5 ToolHM
2 4 ServoM
3 2 ServoM
4 (4 ServoM, 2 ServoM, 5 ToolHM, 3 ServoM)

ID: 0 Data: 1 3 S e r v o M Time: 754

13ServoM

MSG ID(HEX) 00 MSG DATA[0]...DATA[7] 00 00 00 00 00 00 00 00

Figure 67: GUI Showing Numbering Sequence

Once the user enters in the physical configuration details, the control system then re-assigns the initial data downloaded from the modules to the new classes that are generated as shown in Figure 68. Unlike the initial temporary classes which did not know where and how the modules link mechanically, these new classes are specific and know the physical configuration details. These new classes are used as part of the control algorithms that have been implemented.

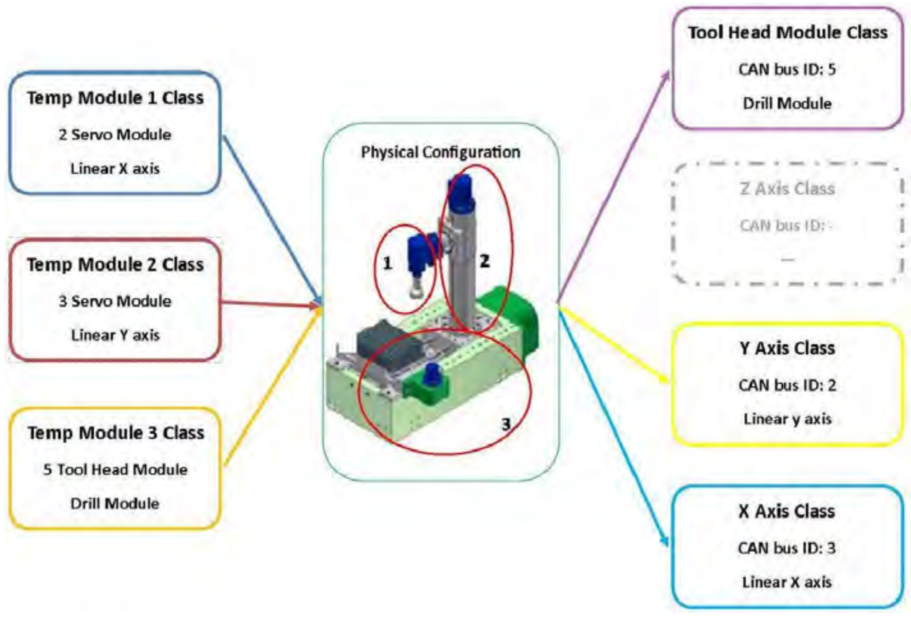


Figure 68: Illustration of the Assigning of Classes

Sample code for the physical configuration and set up of the MRMT can be found in Appendix J, where the functions of the buttons in Figure 68 are implemented. The code illustrates how the OACS responds based on what hardware modules are detected and what physical configuration the user enters. Furthermore since the data is temporarily stored until the physical configuration is entered, the code shows how the data is copied across to new classes based on the user entered configuration.

Knowing the physical configuration and mechanical connections of the modules, the host PC then downloads the specific data information that is required for control purposes from each distributed module.

The system as implemented has both the ability to scan the bus on start-up as well as live time readjust the control variables for the algorithms. The advantage of such a system is that if a new module is added on, whether it may be slower or faster in control speeds to the existing modules, the system caters for the lowest common denominator and uses that to determine the overall system performance. This then ensure that regardless of the module added onto the system, the system is able to function correctly.

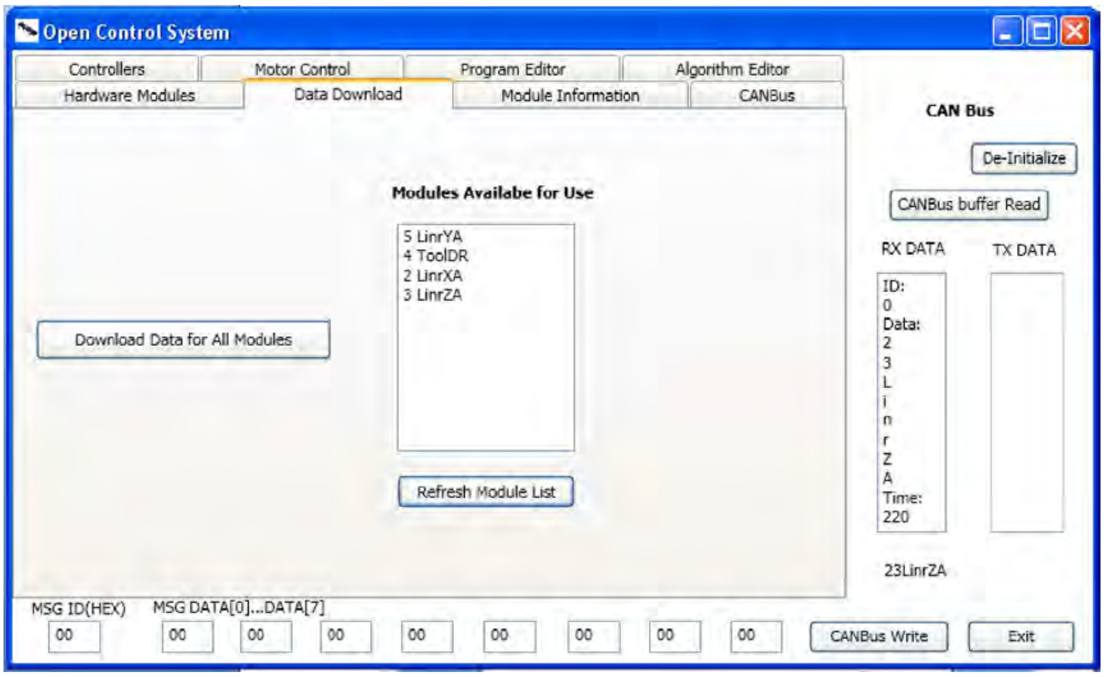
8.5 Data Downloading and Module Information

The OACS conducts a data download from each of the connected distributed modules and requests the following data from each module:

- Type of Module:
  - Axis:
    - Translational Axis
    - Rotary Axis
  - Tool:
    - Drill
- Accuracy;

- Axis limitations and movement range;
- Resolutions;
- Motor speeds;
- Transformation matrix.

Each of these are crucial for ensuring correct operation of the control algorithms. Figure 69 shows an example when data has been downloaded. Appendix K shows sample code for the data download tab where the functions of the buttons in Figure 69 are presented and also covering how the OACS sends out commands to the distributed modules based on the user input.



5 LinrYA
4 ToolDR
2 LinrXA
3 LinrZA

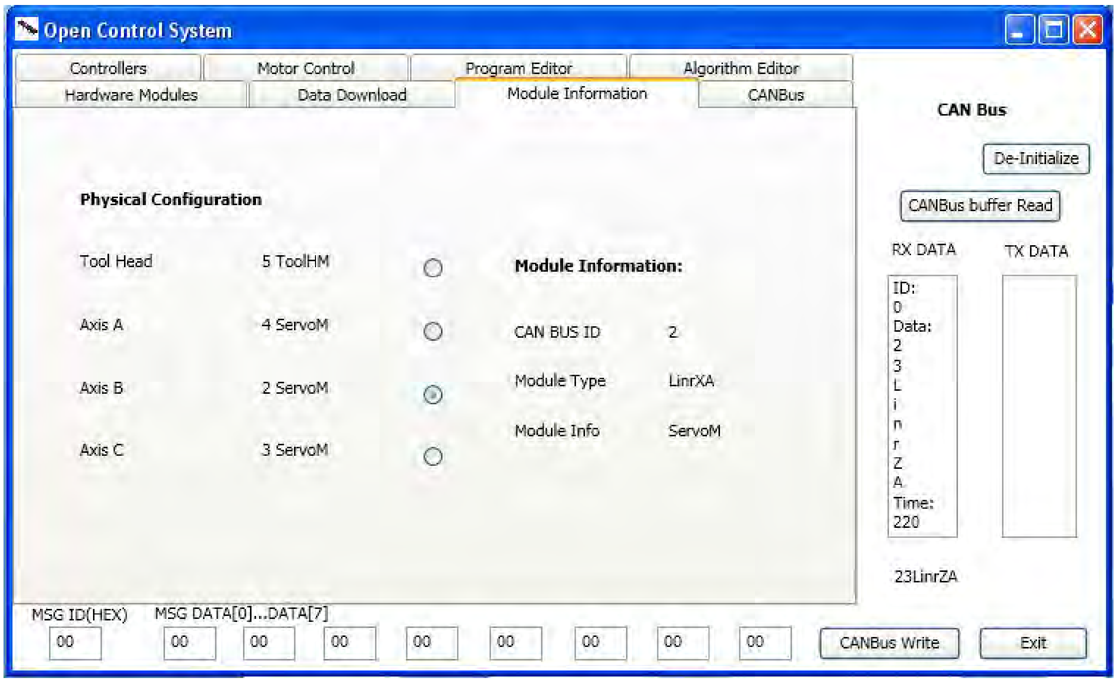
ID:
0
Data:
2
3
L
i
n
r
Z
A
Time:
220

23LinrZA

00 00 00 00 00 00 00 00 00

Figure 69: GUI Data Downloads

Once this data has been downloaded for each distributed module, the system automatically stores the downloaded data in the corresponding software class. Thereafter, if the user requires, the respective data for each module can be displayed. An example of the module information tab is shown in Figure 70. This module information features allow users to conduct verification and assists in diagnostics and debugging to verify that the correct modules are connected and the correct data has been downloaded.



Physical Configuration

Tool Head	5 ToolHM	○
Axis A	4 ServoM	○
Axis B	2 ServoM	◉
Axis C	3 ServoM	○

Module Information:

CAN BUS ID	2
Module Type	LinrXA
Module Info	ServoM

RX DATA: ID: 0 Data: 2 3 L i n r Z A Time: 220

TX DATA:

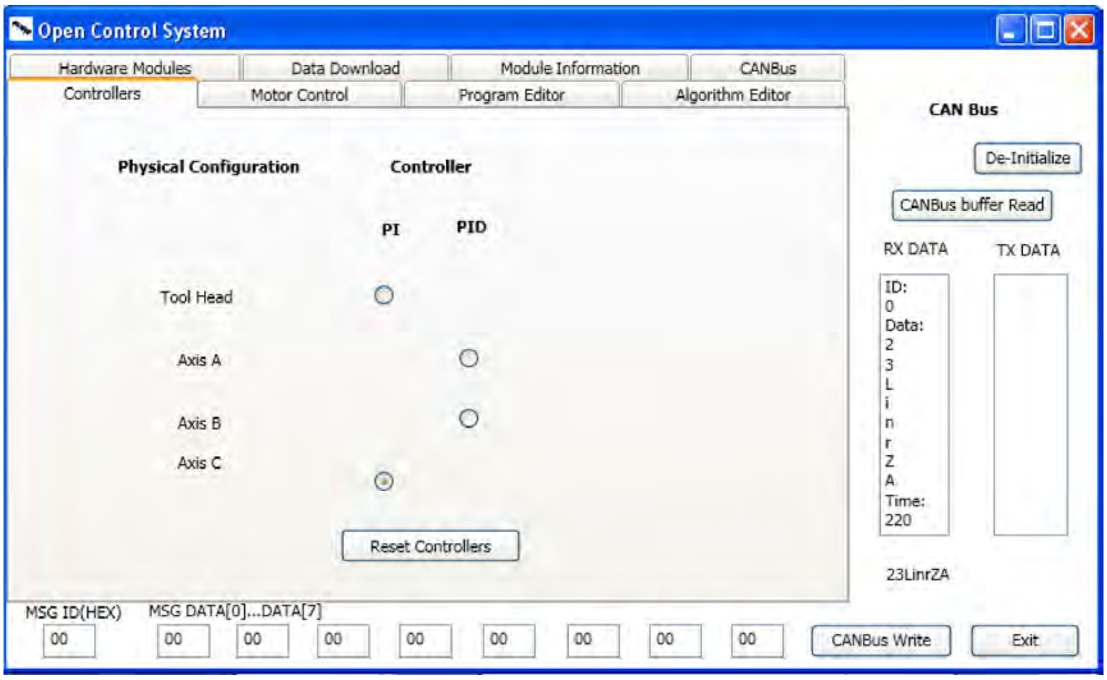
23LinrZA

MSG ID(HEX): 00
MSG DATA[0]...DATA[7]: 00 00 00 00 00 00 00 00

Figure 70: GUI Module Information

8.6 *Contoller Selection*

One of the requirements of the OACS is to allow user customization. One of the methods implemented to allow user customization is to allow users a choice of control algorithms. The OACS presents the user with two choices for control algorithms. This is shown in Figure 71, the choices being: PID or PI control. These two controllers were chosen to show that a user can have a customized solution. However the user may even write their own controller and add it to the OACS if desired.



Physical Configuration — Controller

	PI	PID
Tool Head	○	
Axis A		○
Axis B		○
Axis C	◉	

RX DATA: ID: 0 Data: 2 3 L i n r Z A Time: 220

TX DATA:

23LinrZA

MSG ID(HEX): 00
MSG DATA[0]...DATA[7]: 00 00 00 00 00 00 00 00

Figure 71: GUI Controller Selection

Once the user has chosen his preferred control algorithm, the OACS updates the corresponding software class or module, and in turn sends out the required data to the corresponding distributed module. If required, the OACS can be updated with additional control algorithms, and the end user can then be presented with additional control algorithm choices.

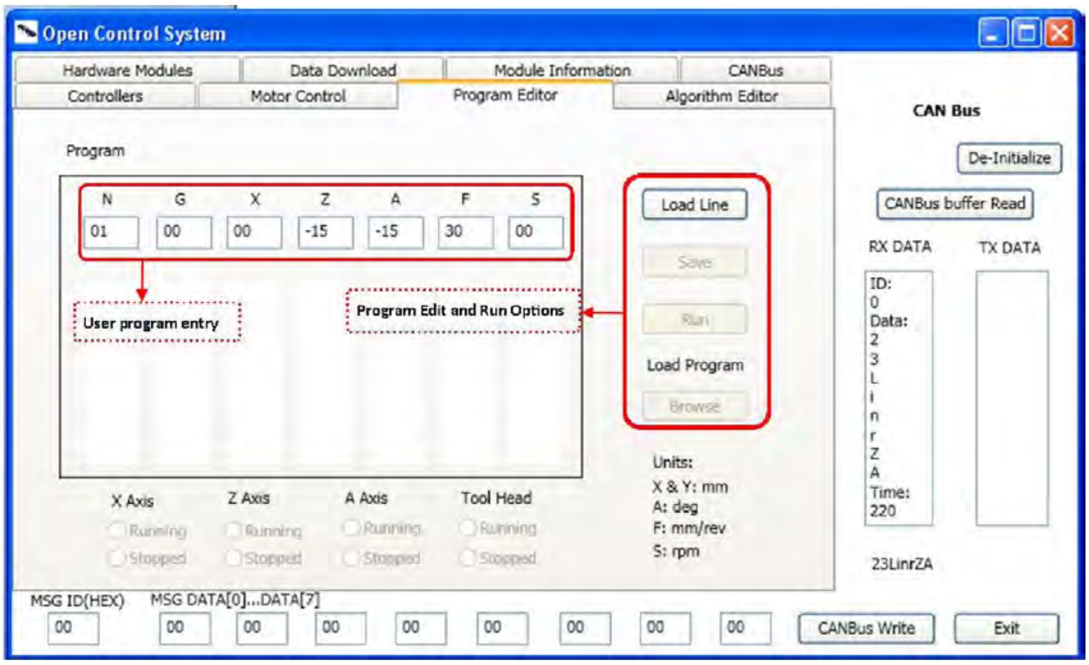
8.7 *User Programming*

A critical aspect of OA platform is to allow end users the ability to: program, edit and write custom applications depending on their machining requirements and the MRMT available. The inclusion of software modularity and reconfigurability, from the outset of the design and development of the control system, has allowed for the addition of methods and functions for user specific customization of the executed programs. The customization methods have been included in three different ways:

- Program Editor
- Algorithm Editor
- Performance Evaluation and Tuning

An end user has the option of writing and inputting their own program using a combination of M and G word functions. A reduced set of M and G word functions has been discussed in Section 7.3. The options for the user to enter the: program, limits and acceptable inputs, has been discussed in Section 7.3.

The program that was entered is saved and compiled, and if a successful compiling flag is received, the user is able to run the program by selecting the run command button. Figure 72 shows the program editor window that the user can write, edit, compile and run programs with. Users can also save and load programs in a program database when required, by simply clicking the corresponding screen commands.



N	G	X	Z	A	F	S
01	00	00	-15	-15	30	00

Units:
X & Y: mm
A: deg
F: mm/rev
S: rpm

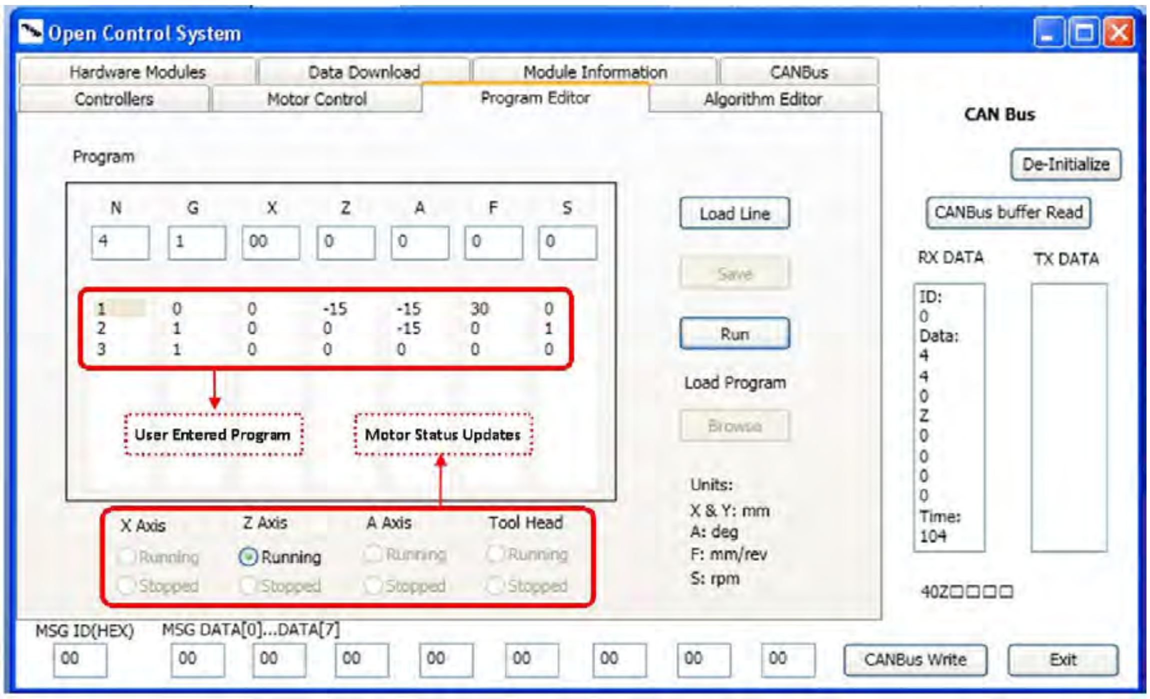
ID:
0
Data:
2
3
L
i
n
r
Z
A
Time:
220

23LinrZA

MSG ID(HEX)	MSG DATA[0]...DATA[7]							
00	00	00	00	00	00	00	00	00

Figure 72: Program Editor

When the user entered program is run, the OACS sequentially steps through the program line by line, transmitting commands to the distributed module, updating the control commands transmitted as completed status messages are received from the modules, and updating the required fields of the GUI as the received data is decoded. An example of the program editor window is shown in Figure 73, where the program is currently executing line 1 of the program. As can be seen, the status of the Z axis is set to "running" indicating that the MRMT is still executing the respective program.



N	G	X	Z	A	F	S
4	1	00	0	0	0	0
1	0	0	-15	-15	30	0
2	1	0	0	-15	0	1
3	1	0	0	0	0	0

X Axis: Running / Stopped; Z Axis: [✓ Running] / Stopped; A Axis: Running / Stopped; Tool Head: Running / Stopped

Units:
X & Y: mm
A: deg
F: mm/rev
S: rpm

RX DATA:
ID:
0
Data:
4
4
0
Z
0
0
0
0
Time:
104

40Z□□□□

MSG ID(HEX): 00
MSG DATA[0]...DATA[7]: 00 00 00 00 00 00 00 00

Figure 73: Program Editor with User Entered Program

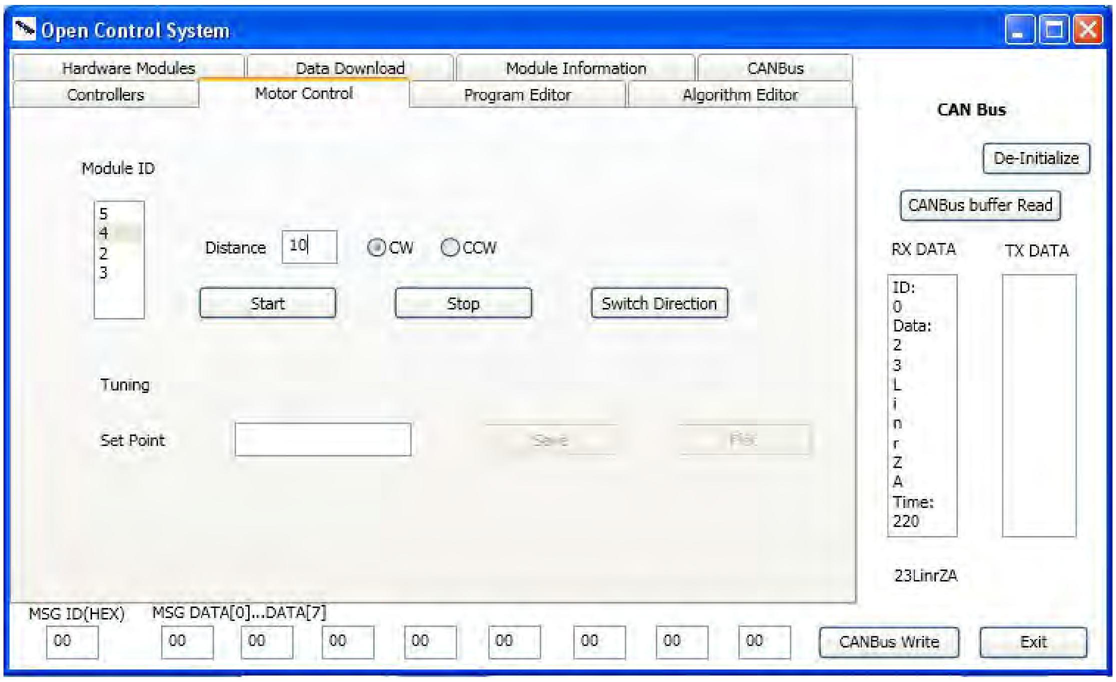
8.8 Motor Control, Debugging and Performance Evaluation

The motor control tab allows the end user to debug the MRMT and could help to assist in diagnosing problems on axes. The end user has the option of individually controlling each axis, to a dedicated set point, a distance of just a movement in either direction. Figure 74, shows the GUI motor control tab, where the user has selected to move the distributed module with ID 4 a distance of 10 mm clockwise.

Sample code for the motor control tab is attached on Appendix L. Once programs are written, compiled and run, the user may wish to evaluate the performance of the control algorithms and their custom written programs. In order to evaluate the performance, the user can track the response of the control algorithms by plotting the feedback response.

The performance evaluation process is started by: selecting a distributed module to test, entering a set point for the axis and starting the MRMT. Once the set point has been reached, the feedback data is saved and the user has the option of plotting the response. Figure 74 shows the motor control screen presented to the user for performance evaluation.

The OACS has proven that an open system can allow end user customization throughout the life time of the MRMT by allowing access to change and modify performance parameters such as controller gains. Similarly, this can be up scaled to include additional end user customization and reconfiguration options such as allowing users to change the interpolation times, cycle refresh times, PID sample times linear interpolation intervals, acceleration and deceleration multiplier values and others.



Module ID: 5 4 2 3

Distance 10 (● CW) ○ CCW

Set Point

ID: 0 Data: 2 3 L i n r Z A Time: 220

23LinrZA

MSG ID(HEX) 00

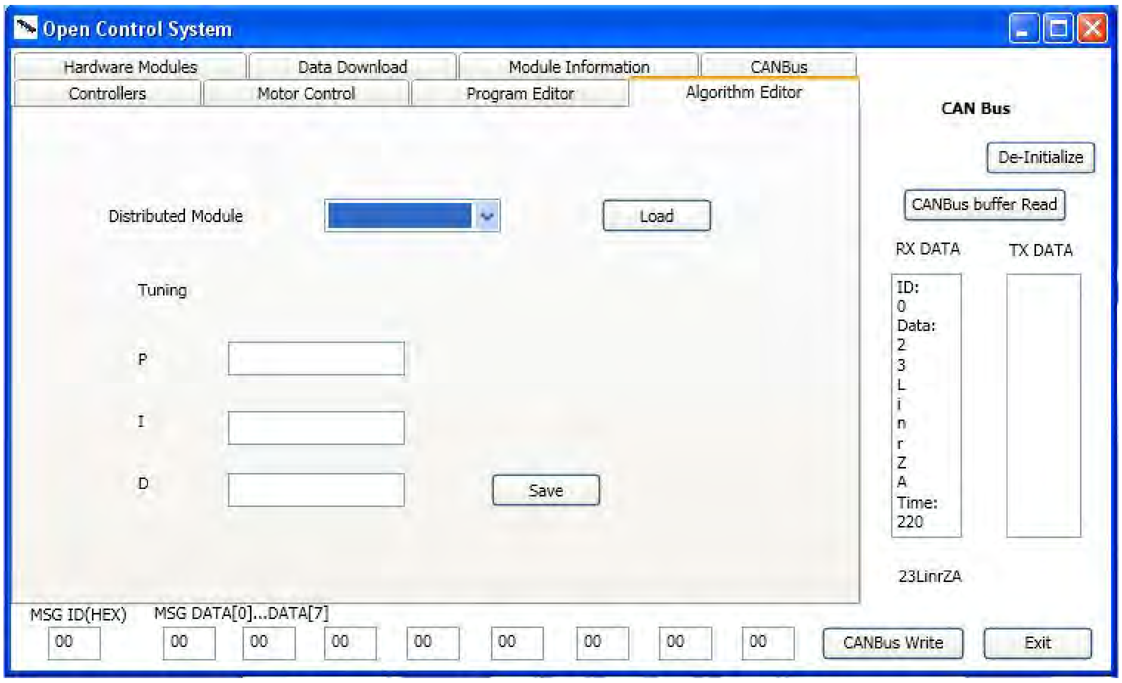
MSG DATA[0]...DATA[7] 00 00 00 00 00 00 00 00

Figure 74: GUI Motor Control and Debugging

8.9 *Algorithm Editing and Tuning*

After analysis of the controller response, and evaluating the performance from the motor control tab, the user can access the algorithm editor screen to tune the controller. By selecting the distributed module, the user can edit or change to tuning parameters of the controller associated with that distributed module.

When the user loads the new tuning parameters, the OACS updates the corresponding software module or class, and in turn transmit the updated parameters to the respective distributed module. The algorithm editor tab, where tuning parameter editing and update can be achieved, is presented in Figure 75. The PI or PID controller performance can be assessed and tuned as per the Ziegler Nichols tuning method, or by manual tuning, covered in Section 7.2.



RX DATA

```
ID:
0
Data:
2
3
L
i
n
r
Z
A
Time:
220
```

TX DATA

23LinrZA

MSG ID(HEX)	MSG DATA[0]...DATA[7]							
00	00	00	00	00	00	00	00	00

Figure 75: GUI Algorithm Editor

8.10. *Distributed Module Software*

The distributed modules were responsible for interfacing the mechanical modules and the associated sensors. As covered in Section 6.3, the distributed modules were developed using multiple microcontrollers to demonstrate the openness of the OACS. The programming environments for each of the distributed modules differ depending on the type of microcontrollers. The following programming environments were used:

- FEZ Panda 2;
  - Microsoft Visual C#
    - With Microsoft .Net micro Framework SDK
    - FEZ Panda 2 libraries
- Arduino Uno;
  - Arduino 1.0.1
    - With CAN_BUS Shield Libraries
- chipKIT Max 32;
  - MPIDE
    - With chipKIT network Shield libraries for CAN bus

The programming of distributed modules, despite being on multiple microcontrollers, all followed the generic flowchart shown in Figure 76. Depending on the programming environment, the microcontroller features, and interfacing circuitry, the actual programs on the microcontrollers may vary from the generic flowchart which forms part of end user customization and flexibility.

The flexibility in microcontroller choice is part of the end user customization to ensure that despite the differences between distributed modules, with a small customization on the modules, the complete system functionality is maintained, provided that the interface and data transfer between the tiers is consistent and standardised.

8.11 Chapter Summary

Chapter 8 has covered a discussion on the GUI of the OACS. The various functions of the OACS that the user is able to access has been discussed and the value add of each function highlighted. Furthermore a flowchart of sequence of events that the user is expected to follow is presented which closely relates to the function of the OACS. Finally the distributed module software is also covered, with a flowchart to illustrate what the distributed modules are required to manage.

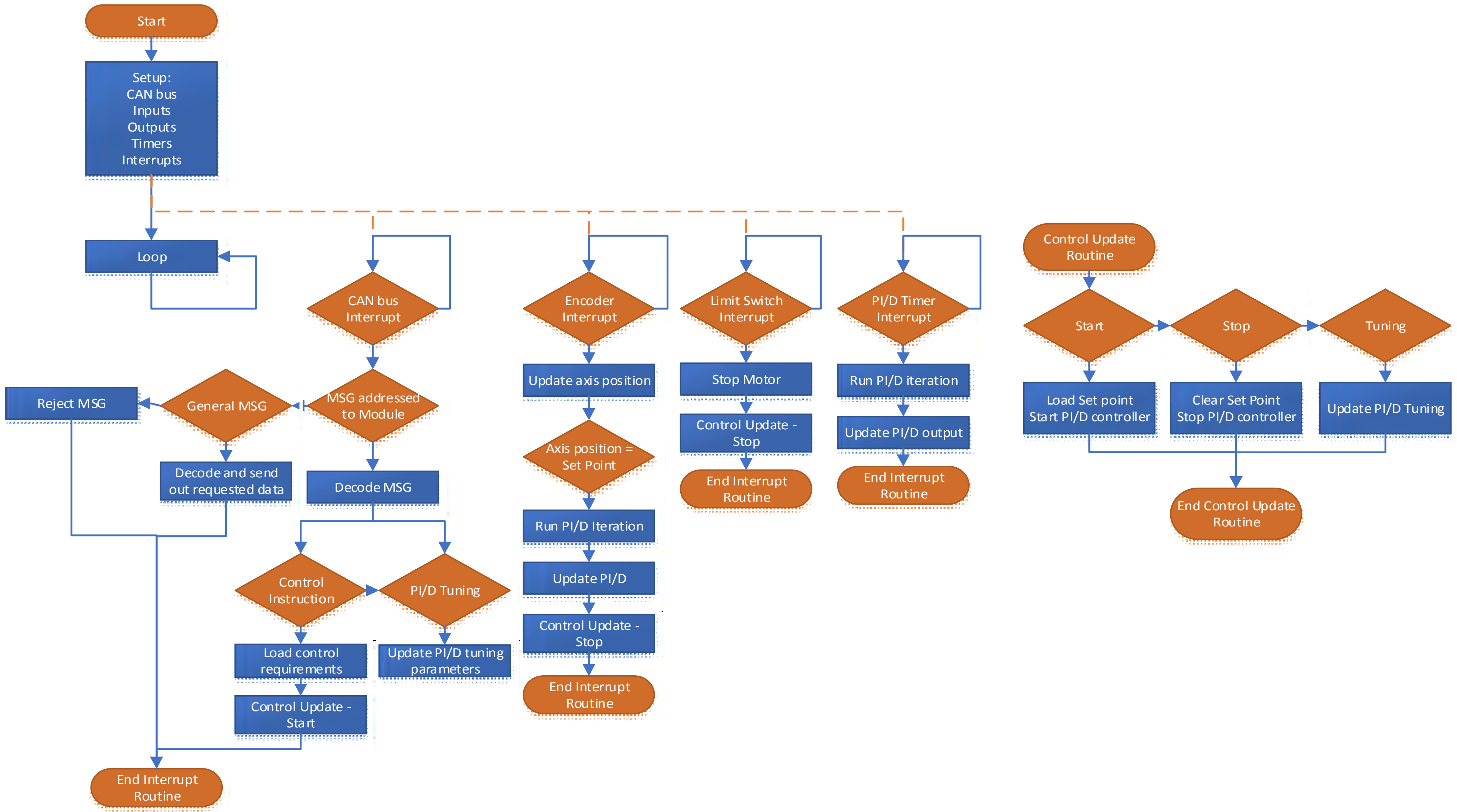


Figure 76: Microcontroller Software Flowchart

9. System Testing and Performance

The performance of the researched, designed and developed OACS on the existing MRMT is firstly evaluated in terms of accuracy, repeatability, power usage and vibration monitoring for machining and control. Additionally the CPU load, reconfiguration times and response times based on a multiple microcontroller implementation is also discussed to evaluate the success of the developed system.

9.1. MRMT Setup

To conduct tests for performance on the MRMT and the OACS, the MRMT was set up using the four mechanical and electronic modules namely: two linear axes, one rotary axis and the spindle module. The MRMT was configured as shown in Figure 77 for the tests.

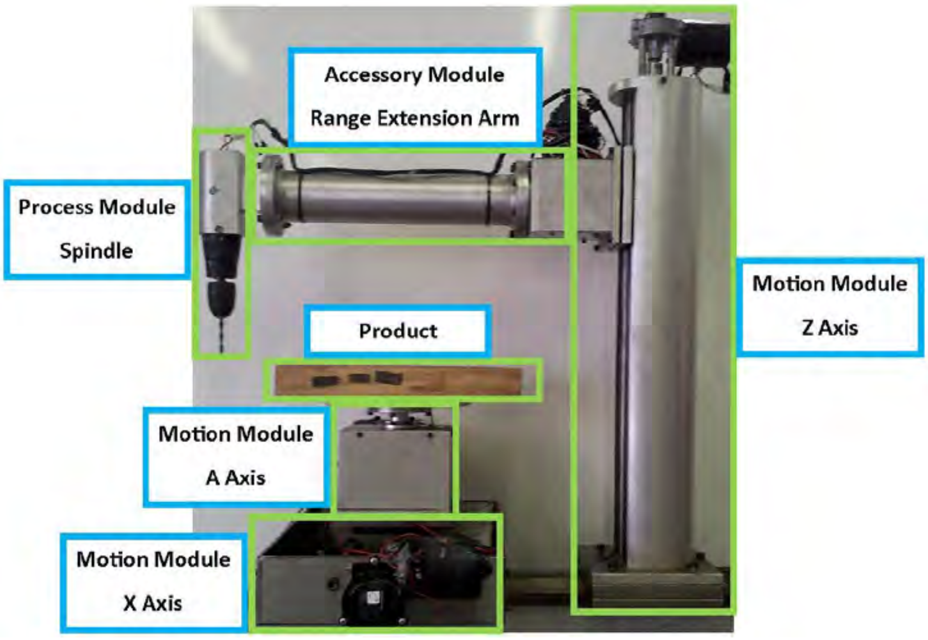


Figure 77: MRMT Configuration for Testing

The distributed modules were each connected to a mechanical axis, thus ensuring that the associated electronics such as: encoder input circuitry or limit switch input circuitry, for that axis, were available on that distributed module. The microcontroller for each distributed module was mounted on a single board with a Vera board, which contained all the circuitry to support the external interfaces, as shown in Figure 78, Figure 79, Figure 80 and Figure 81. The Figures show pictures of each of the distributed modules as constructed, with labels indicating key circuitry on each of the modules.

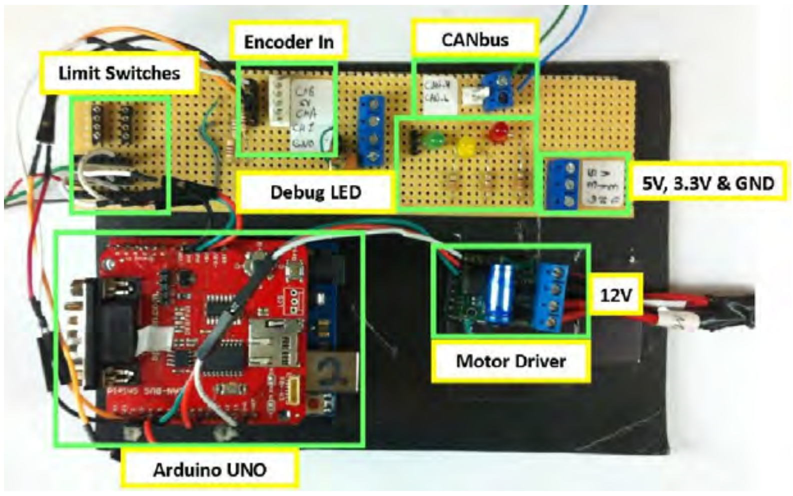


Figure 78: Distributed Module - Arduino UNO (Linear Axis Z)

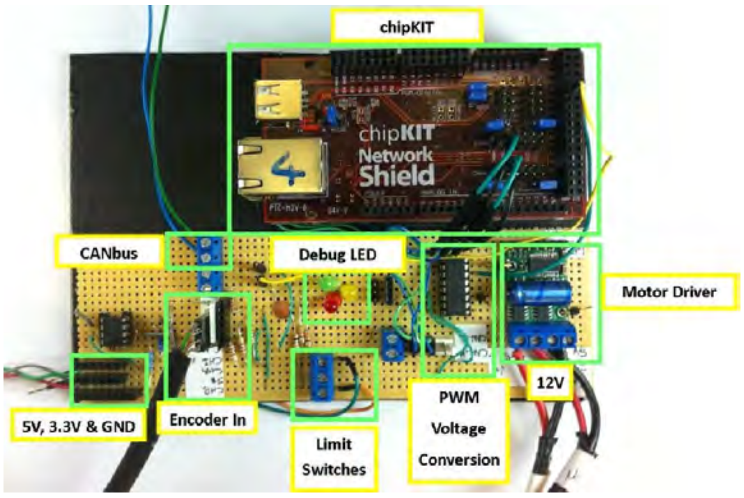


Figure 79: Distributed Module – chipKIT (Linear Axis X)

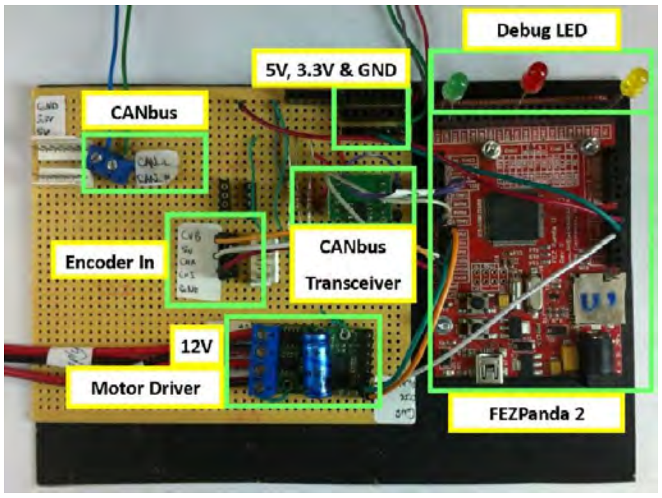


Figure 80: Distributed Module – FEZ Panda 2 (Rotary Axis A)

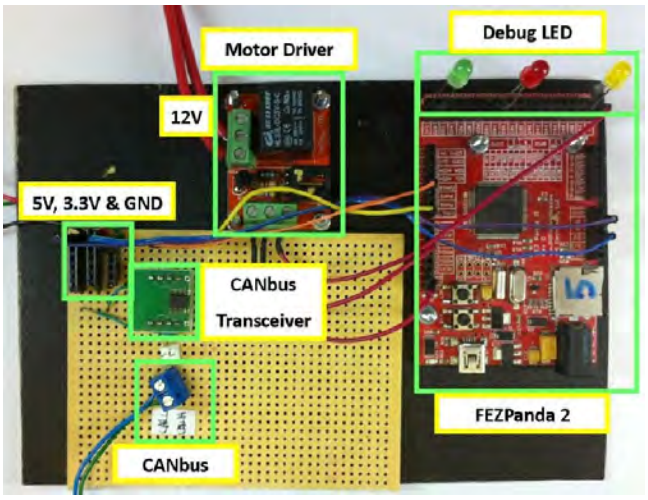


Figure 81: Distributed Module – FEZ Panda 2 (Spindle Module)

9.2. Reconfiguration Times

The mechanical modules of the MRMT were set up with two different configurations, one with three modules and the other with four modules. For each of the aforementioned scenarios, the time for the distributed module reconfiguration, followed by the OACS reconfiguration, which included entering of a generic user program, was captured. In order to compare the results to a reference, the tests were conducted, firstly by a user familiar with the OACS, and thereafter by a new user who had no experience on the OACS. The tests were conducted five times and the averaged results are summarised in the Figure 82. The full set of results can be found in Appendix M.

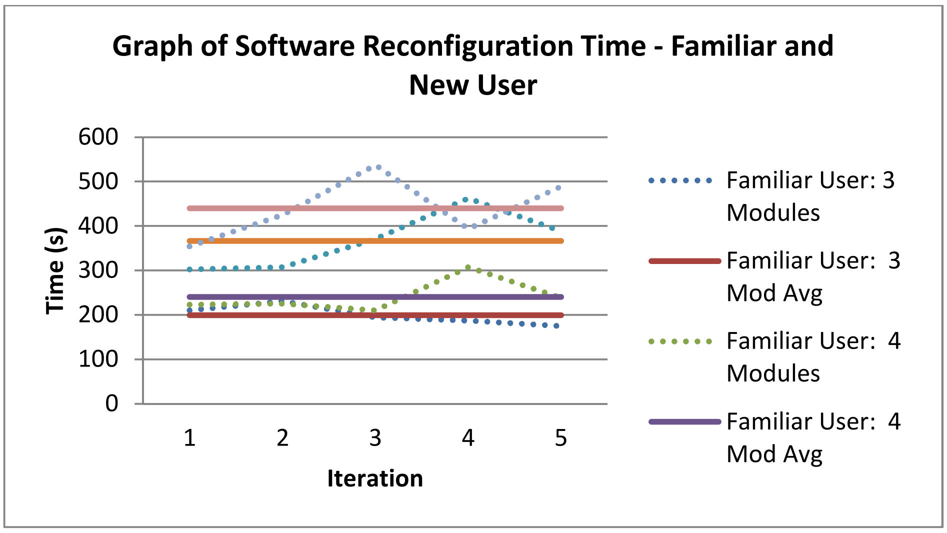


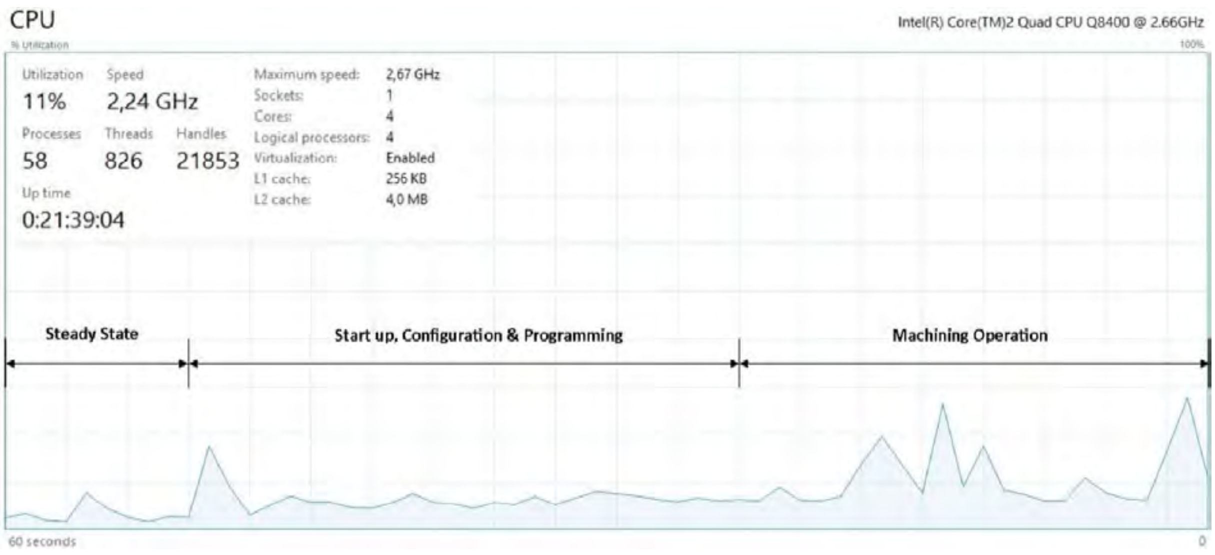
Figure 82: Graph of Software Reconfiguration Time

From these results, the following can be seen:

- Reconfiguration and setup of the OACS is quicker when conducted by a user familiar to the system;
- New users took approximately 1.75 times longer than familiar users to reconfigure the OACS.

9.3. PC Load

A windows performance monitor was used to test the load of the OACS on the PC. The performance monitor allows the CPU and PC memory usage to be monitored. Figure 83 is a snapshot of the CPU utilization during operation of the OACS.


CPU

Intel(R) Core(TM)2 Quad CPU Q8400 @ 2.66GHz

Utilization	Speed		Maximum speed:	2,67 GHz
11%	2,24 GHz		Sockets:	1
			Cores:	4
Processes	Threads	Handles	Logical processors:	4
58	826	21853	Virtualization:	Enabled
Up time			L1 cache:	256 KB
0:21:39:04			L2 cache:	4,0 MB

Figure 83: CPU Load During OACS Operation

From this data, the following can be seen:

- During steady state, the average CPU usage is less than 4%;
- CPU load increases as the OACS program is started and the MRMT is: set up, configured and programmed;
- The demand on the CPU increases significantly during machining. This is due to the interpolation, acceleration and deceleration as well as the control routines are run;
- The maximum CPU usage recorded is 38%.

9.4. Acceleration and Deceleration Control Example

The acceleration and deceleration routine covered in Chapter 7 manipulates the interpolated data to accelerate the axes on start-up, and decelerate the axis when it reaches its destination point in order to: to minimise overshoot, smoothen the axis movement, and prevents mechanical shock and damage to the MRMT. The S-Shape convolution method had been selected and implemented for the OACS. The corresponding test data is attached in Appendix N, and the simulation graphs are presented in Figure 84.

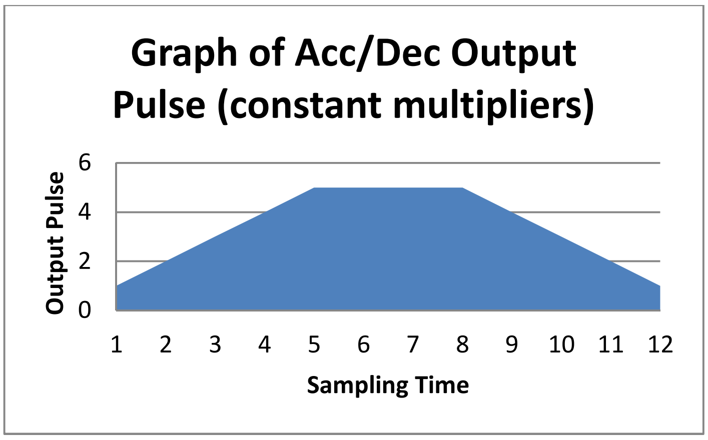


Figure 84: Graph of Acc/Dec Output Pulse

As seen in Figure 84, with constant multipliers, the output follows a linear relationship. However, depending on the electromechanical components, a response curve that follows an S shape maybe desirable for the end user. In order to achieve this S-shape, the same routine can be used with different multiplier values. The example that follows changes the multiplier vales to ***k* = [1,2,3,2,1]**. The resulting plot shown in Figure 85 has a more distinct S-Shape as desired for the smoother acceleration and deceleration control of the axis which relates to the speeding up or holding back of the axis on movement.

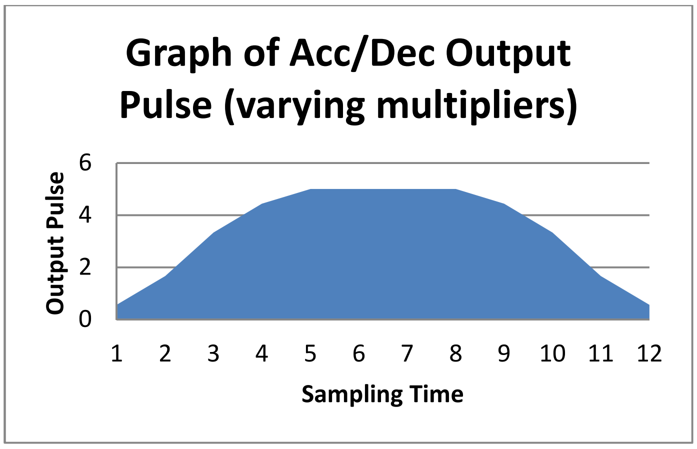


Figure 85: Graph of Acc/Dec Output Pulse

9.5. *Accuracy and Repeatability*

The OACS was designed and built with the aim of satisfying certain performance characteristics. Table 14 summarises the design specifications for each of the MRMT axes which is used to evaluate the performance of the OACS.

Table 14: Summary of MRMT Design Specifications

	Control Resolution	Worst Accuracy	Worst Repeatability	Minimum Speed
X – Axis	$5.9x10^{-3}$	1 mm	±0.5 mm	100 mm/min
Z – Axis	$5.9x10^{-3}$	1 mm	±0.5 mm	100 mm/min
A – Axis	0.015^{o}	1.5^{o}	$\pm1^{o}$	360^{o}/min

To evaluate the performance of the OACS in terms of accuracy and repeatability, a set of test results was required. These tests results are then compared to the aforementioned design specifications, and based on the comparison, the OACS performance was evaluated. From Table 14, the expected counts per unit can be determined via inverting the resolution as shown below:

- X Axis – 169 counts per mm;
- Z Axis – 169 counts per mm;
- A Axis – 67 counts per degree.

These are used to check the accuracy of the test results by comparing the actual encoder count to the expected count. The MRMT was tested by commanding each axis to execute a certain movement and the results from the movements were captured. To ensure correct evaluation, each test was conducted several times so that the data could be averaged and analysed. The encoder counts for each of the axes were downloaded, from each of the distributed modules, after the move commands were executed. A summary of this data is presented in Figure 86, Figure 87 and Figure 88. In these figures the average counts, as well as the actual counts, for several iterations are all graphed. The complete data set is attached in Appendix O.

Table 15 compares the average counts with the maximum deviations for all axes. From this data, the repeatability in terms of in terms of worst case deviation from set points is computed.

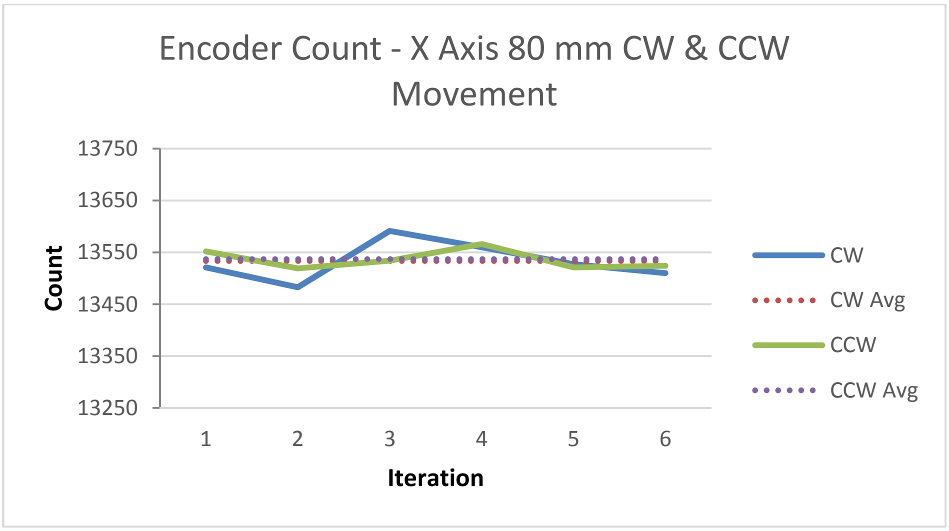


Figure 86: X Axis Encoder Counts

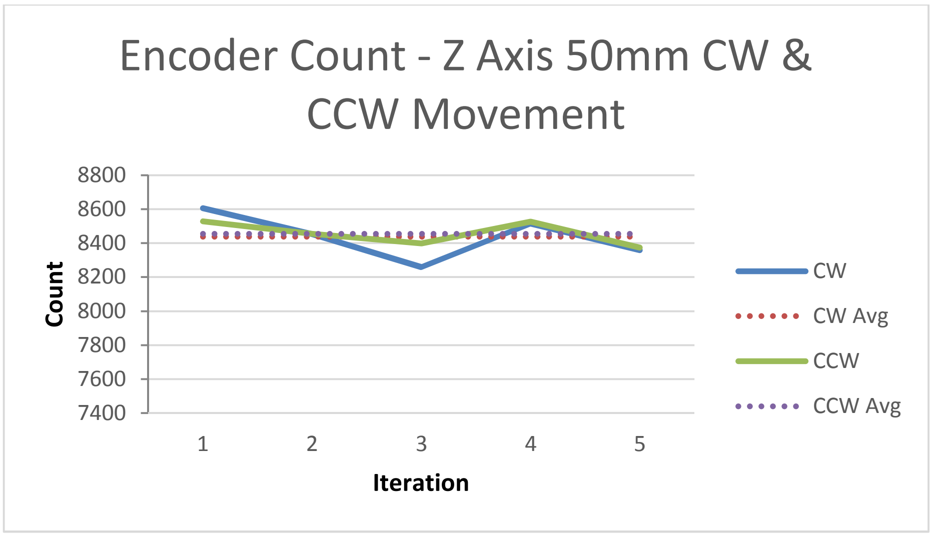


Figure 87: Z Axis Encoder Counts

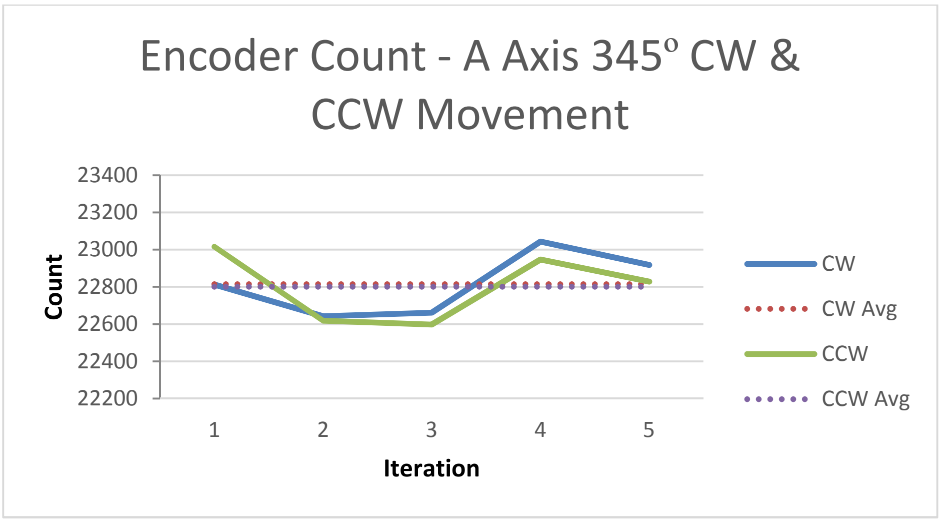


Figure 88: A Axis Encoder Counts

Table 15: Summary of Test Results for Repeatability

	X Axis CW	% Variation	Z Axis CW	%Variation	A Axis CW	%Variation
Average Count	13533		8438		22815	
Smallest Count	13483	0.36	8259	1.07	22643	0.75
Largest Count	13591	0.42	8605	1.97	23017	0.88
	X axis CCW	**% Variation**	**Z Axis CCW**	**%Variation**	**A Axis CCW**	**%Variation**
Average Count	13537		8456		22801	
Smallest Count	13519	0.13	8375	0.82	22597	0.89
Largest Count	13566	0.21	8529	0.99	23016	0.94

To test the accuracy of the system, the smallest and largest encoder counts are compared to the expected count based on the module control resolution. Through this computation, the worst case accuracy is computed for test data. Table 16 summarizes the accuracy results for the translational axes.

Table 16: Summary of Accuracy Test Results for Translational Axes

	X Axis CW	**Deviation**	**Z Axis CW**	**Deviation**
Expected Count	**13520**		**8540**	
Difference on Lowest Count	37	0.22 mm	191	1.13 mm
Difference on Highest Count	71	0.42 mm	155	0.92 mm
	X axis CCW	**Deviation**	**Z Axis CCW**	**Deviation**
Expected Count	**13520**		**8540**	
Difference on Lowest Count	1	0.005 mm	75	0.44 mm
Difference on Highest Count	46	0.27 mm	79	0.47 mm

The following can be deduced from these comparisons:

- All Axes show similar trends with the average encoder counts for CW and CCW rotations almost the same;
- The X-Axis has the best performance with worst case deviation of 0.42 mm;
- A Axis has best repeatability % variations of less than 0.21% for CCW rotations and less than 0.42% for CW rotation;
- The Z-Axis CCW rotation is less than the acceptable 1% variation, though the CW rotations is between 1 and 2 % in terms of repeatability;
- The Z Axis accuracy is within the acceptable 1 mm for the three of the four worst case readings;
- Both linear axes produce 169 counts per mm;
- This corresponds to the control resolution of $5.9x10^{-3}$ mm.

Similar tests for the rotary A axis were conducted and revealed errors between 1-7% and worst case repeatability of less than 1%. Further to this, measurement tests revealed that the A Axis had actually rotated approximately 22 degrees over the target point of 360 degrees. Since the controller uses the feedback from the encoders to determine the current axis position, the assumption was that either the encoder was not outputting all signals or the microcontroller was not detecting these signals. Initially, the encoders were changed and the interfacing circuitry checked but this did not reveal any change in result. A further test was conducted where the axis was manually moved to a measured point and a check on the encoder counts revealed that the microcontroller was actually only recording 63 pulses per degree and not 67 as expected. This proved that the microcontroller was in fact missing pulses.

To correct this, the encoder resolution for the A axis was halved to 512 counts per revolution from 1024 counts per revolution. This change would affect the system control resolution by doubling it to 0.03^o and halving the expected count per degree to 33, but would reveal if the microcontroller was in fact missing encoder pulses. Figure 89 and Table 17 summarise the results captured.

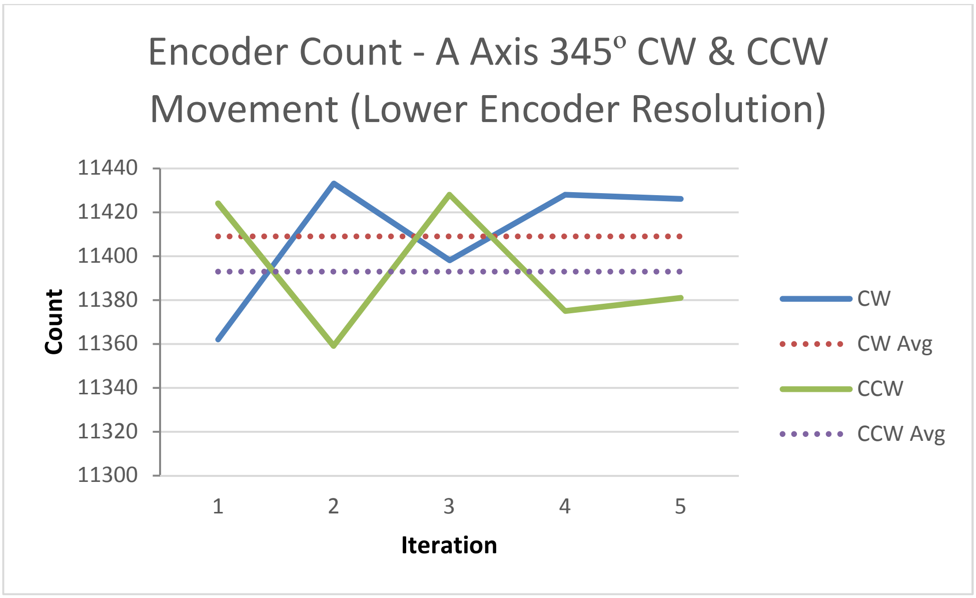


Figure 89: A Axis Encoder Counts (Lower Encoder Resolution)

Table 17: Summary of Test Results for Accuracy and Repeatability for A Axis

	A Axis CCW	% Variation	A Axis CW	% Variation
Average Count	**11393**		**11409**	
Smallest Count	11359	0.29	11362	0.41
Largest Count	11428	0.31	11433	0.21
	A Axis CCW	**Deviation**	**A Axis CW**	**Deviation**
Expected Count	**11385**		**11385**	
Difference on Lowest Count	26	0.78 ^{o}	22	0.69 ^{o}
Difference on Highest Count	43	1.3 ^{o}	48	1.45 ^{o}

From this experimental data, the following can be deduced:

- Significant improvement in accuracy of results with the reduction of the encoder resolution;
- Accuracy to within 1.45^o of the expected position, with the specification of 1.5^o;
- System performs repeatable performance with worst case variation of 0.41% for repeatability;
- Clear indication that the FEZ Panda 2 board despite running on 72 MHz cannot manage 1024 interrupts per rotation of the motor.

In order to further test the repeatability of the system, the time taken to complete a command for each axis was recorded using Peak Systems PCAN view, which monitors the data on the bus. To ensure a correct evaluation, each test was conducted several times so that the data could be averaged and analysed. Figure 90 compares the time taken for movement of each axes, in a CW and CCW direction. The average times were calculated from the time the movement command was transmitted from the host PC, till the time the feedback messages was received. This data was monitored and captured using PCAN view, a CAN bus data monitor [42] from PEAK systems.

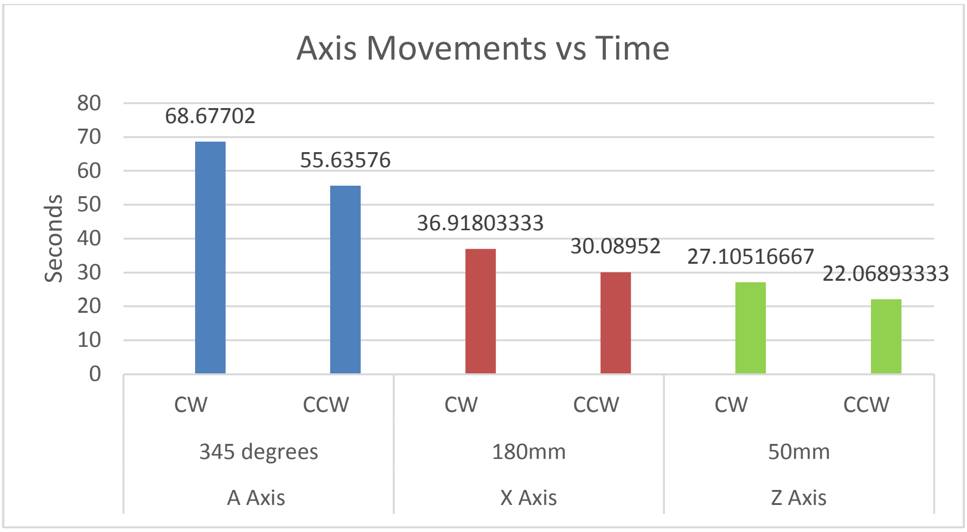


Figure 90: Axis Movement vs Average Time

Table 18: Comparison of Axis Speeds

	X Axis	Z Axis	A Axis
CW Movement	130 mm/min	110 mm/min	301 ^{o}/min
CCW Movement	159 mm/min	135 mm/min	372 ^{o}/min

Table 18, showing average axis speeds, can be derived based on the data captured. From these results the following can then be deduced:

- There is consistency for all axes, where times for CW movement are larger than the corresponding time for CCW movement;
- Analyzing the time graph reveals that CCW time is on average 81.3 % of CW time where:
  - A Axis – 81.01 %;
  - X Axis – 81.50 %;
  - Z Axis – 81.42 %.
- The X and Z axes both satisfy the minimum speed requirements of 100 mm/min;
- The A Axis CCW movement satisfies the minimum speed requirement as per the design specification. However the CW movement fails and does not achieve the minimum required speed of 360 ^{o}/min.

9.6. Vibrations

An accelerometer sensor formed part of the interfacing peripherals of the spindle distributed module to allow machining operations to be monitored. The 3-way accelerometer measured the gravitational accelerations in the X, Y and Z planes, and the system was tested for vibrations in three different scenarios. These are as follows:

- Test 1: To check steady state vibrations as well as vibrations due to machine movement;
- Test 2: To check intensity of vibrations during 20 mm drilling operation;
- Test 3: To check intensity of vibrations during 40 mm drilling operation.

The accelerometer measured gravitational acceleration and outputted an analog signal. This signal is fed through an Analog to Digital Convertor (ADC). The results of the ADC are presented in Figure 91, Figure 92 and Figure 93.

The system uses the 10-bit ADC on the microcontroller allowing 1024 levels on a 5 volt signal. At 3.3V power to the accelerometer, the accelerometer uses a gravitational resolution of $\mathbf{330}\frac{mV}{g}$, and the actual gravitation acceleration can be computed by equation 9.1 [52] where V_{ref} is the reference voltage, V_{norm} the supply voltage to the accelerometer and n the number of the bits on the ADC.

$$Gravitational\ Acceleration = g(V_{ref}\frac{Reading}{n^2} - V_{norm}) \tag{9.1}$$

For each axis, the steady state value, when the accelerometer measured no gravitational acceleration, as well as the maximum and minimum readings taken, are presented in the Table 19. The complete set of data is attached in Appendix P.

Table 19: Range of Accelerometer Readings Captured

	X (mv/g)	Y (mv/g)	Z (mv/g)
Steady State Reading	404	522	499
Maximum Reading	481	566	558
Minimum Reading	378	426	422

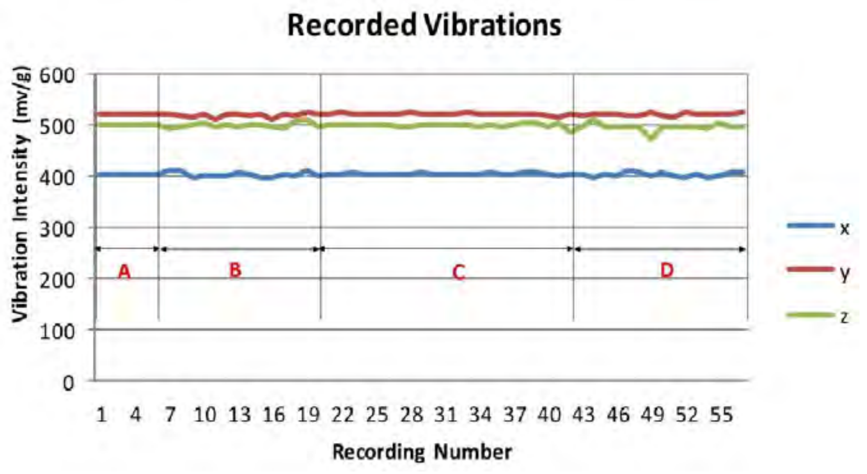


Figure 91: Recorded Vibrations Due to Axis Movement and Drill Operation

Figure 91 is a record of the vibration intensity during the following stages:

- A: No axial or spindle operation, no vibrations recorded;
- B: Only spindle operation(no drilling on product), small vibrations recorded;
- C: Only axial movement, minor vibrations recorded;
- D: Axial and Spindle operation (not on product), recorded vibrations greater than B and C.

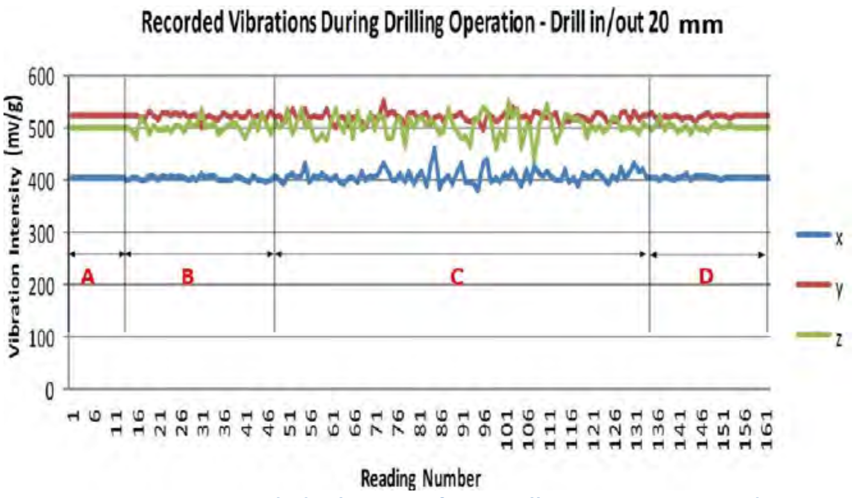


Figure 92: Recorded Vibrations from Drilling 20mm into Product

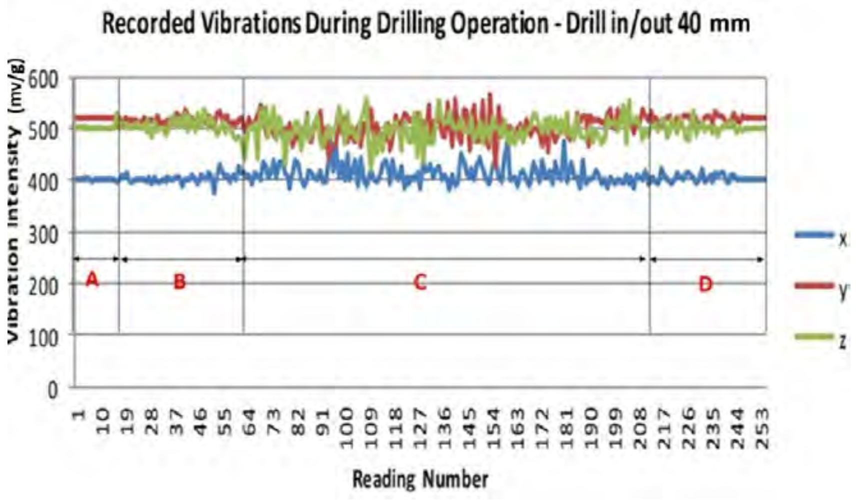


Figure 93: Recorded Vibrations from Drilling 40mm into Product

Figure 92 and Figure 93 are records of the vibration intensity during the following stages:

- A: No axial or spindle operation, no vibrations recorded;
- B: Axial and Spindle operation, movement towards product, small vibrations recorded;
- C: Axial movement and spindle operation(drilling in and out of product;
- D: Axial and Spindle operation, movement away from product, small vibrations recorded.

From Figure 91, Figure 92 and Figure 93, the following can be deduced:

- There is a distinct increase in recorded vibrations on the drill module, from steady state drill operation to drilling on a product;
- From Figure 92, vibrations occur mainly from drill operation, but also from axial movement;
- Vibrations are a combination of MRMT axial movement and drill vibrations;
- As drill depth increases, comparing between 20 mm and 40 mm depth, the vibration intensity increases significantly.

9.7. Power Distribution Network Loading

To evaluate the load on the power distribution network, the current usage and power efficiency of the OACS was analysed. This analysis was based on the current consumption data recorded for several test scenarios. Figure 94, Figure 95 and Figure 96 show the current consumption for the three axial distributed modules, where values for: the CW rotations, CCW rotations and the average current consumption.

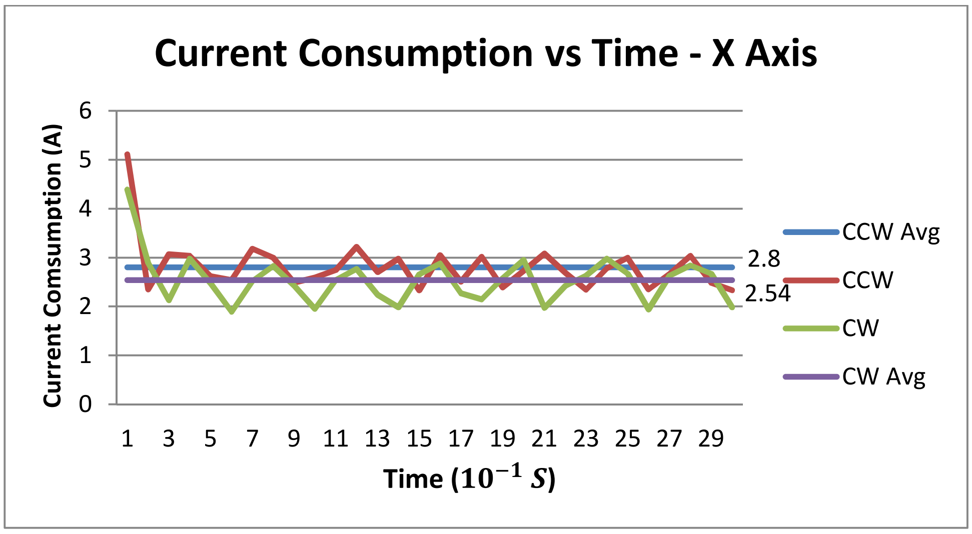


Figure 94: Graph of X Axis Current Consumption

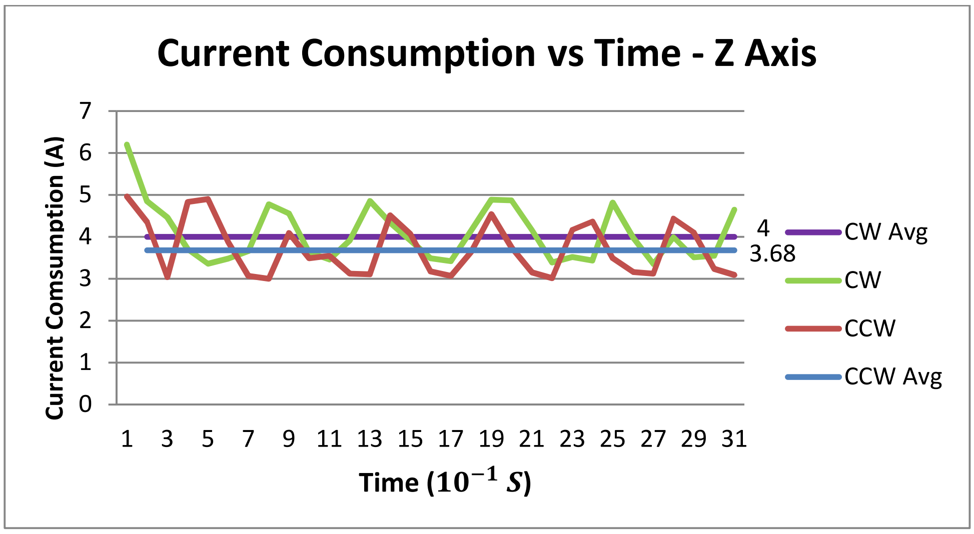


Figure 95: Graph of Z Axis Current Consumption

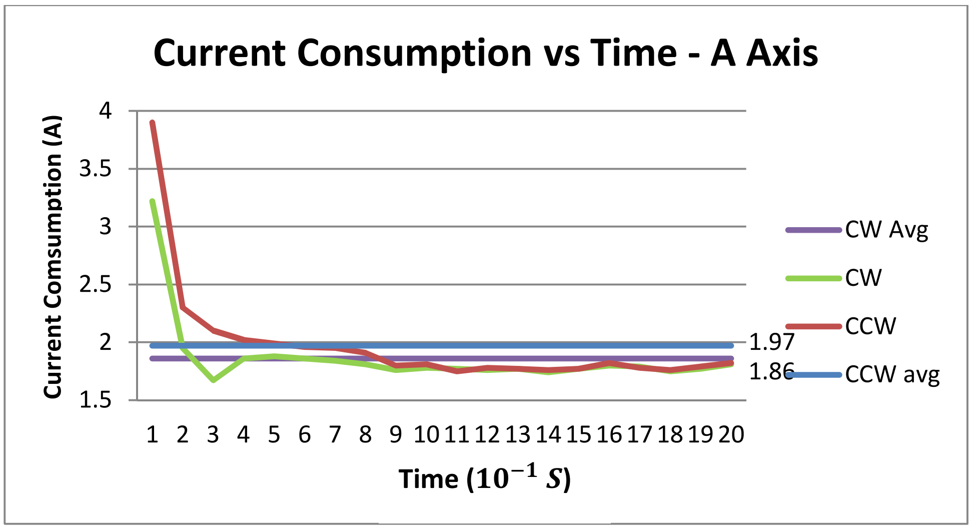


Figure 96: Graph of A Axis Current Consumption

The following is evident from Figure 94, Figure 95 and Figure 96:

- Stall currents for CW rotation on linear axis is greater than stall current for CCW rotations;
- CCW average is greater than the CW average for X and A axes;
- For Z Axis, which is the upright column, the CW average is greater than the CCW average current:
    - CW motion on Z Axis corresponds to an upwards movement and the Z axis has to move up the accessory and spindle module attached to it. Thus due to the added load, a higher current consumption for upward motion as compared to downward motion is expected.
- Distinct sinusoidal-like trend on the current recordings for both translational axes implying increased load during rotation of the motor shaft.

Figure 97 shows the current consumption when the drill is started to steady state operation when no drilling is taking place. On the other hand, Figure 98 shows the current consumption during drilling into a product. A full table of current consumption for the tool module can be found in Appendix Q.

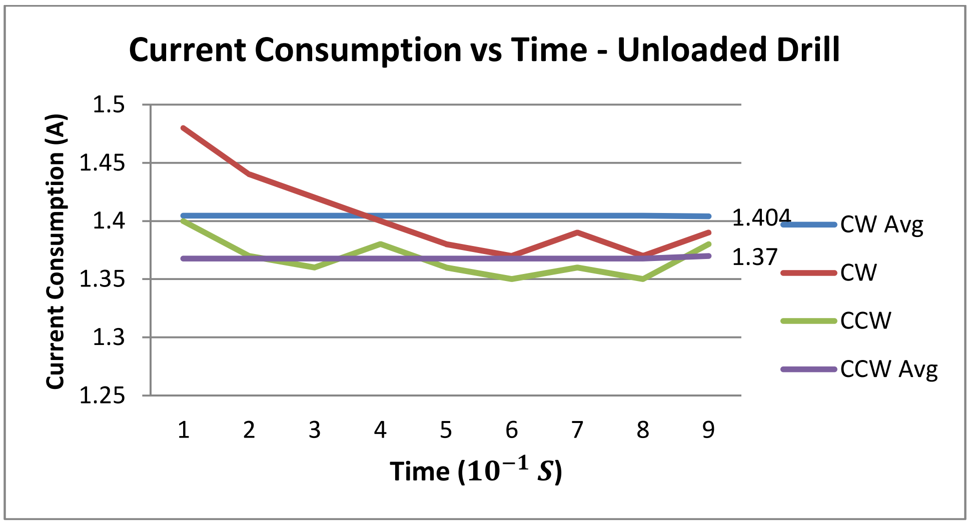


Figure 97: Graph of Unloaded Drill Current Consumption

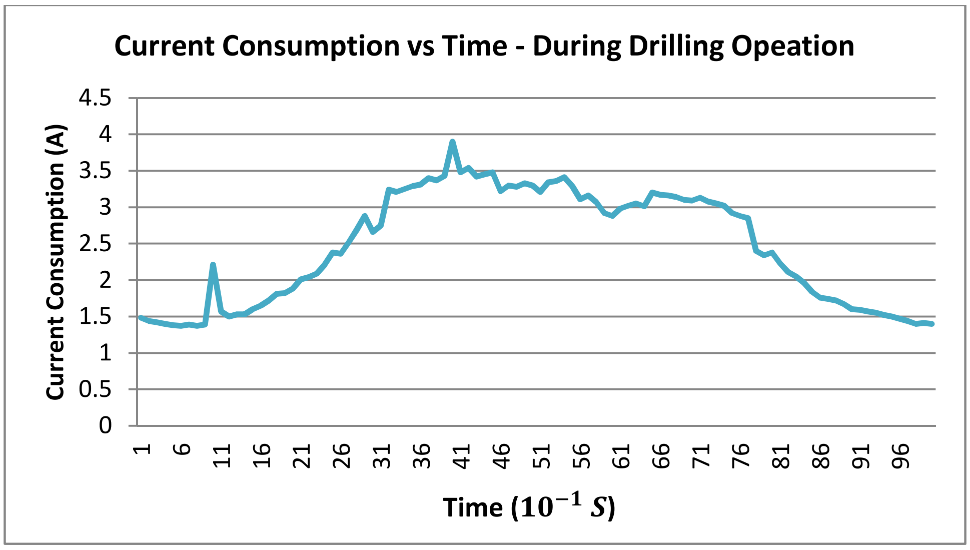


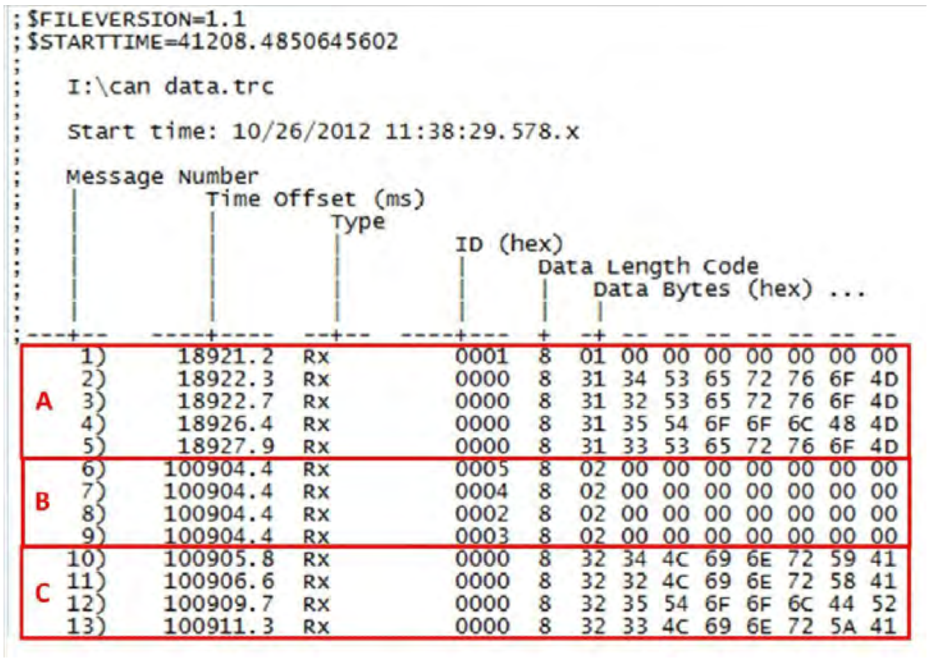
Figure 98: Graph Drill Current Consumption During Operation

The following can be deduced from the data in Figure 97 and Figure 98:

- The first spike correlates to when the drill bit makes first contact with the product, and as the drill depth increases so too does the current required increase;
- In comparison to Figure 93, vibrations during drilling operation, as the drill depth increases there is greater vibration, therefore there is greater resistance increase and thus the current increases towards the middle of the graph.

9.8. Response Times

The OACS has been designed and implemented using multiple microcontrollers, each with different architectures and operating frequencies, as discussed in Chapter 6. To verify if these modules are operating and communicating as required, the communication interface and the microcontroller data exchange is tested. To verify this, the response times of data exchange from the host PC, with each of the distributed modules, has been captured and is presented in Figure 99. PCAN view from Peak Systems, an online CAN bus monitoring and diagnostic tool, was used to monitor the bus activity and bus health.



```
;$FILEVERSION=1.1
;$STARTTIME=41208.4850645602
;
;   I:\can data.trc
;
;   Start time: 10/26/2012 11:38:29.578.x
;
;   Message Number
;   |         Time Offset (ms)
;   |         |        Type
;   |         |        |        ID (hex)
;   |         |        |        |     Data Length Code
;   |         |        |        |     |   Data Bytes (hex) ...
;   |         |        |        |     |   |
;---+--   ----+----  --+--  ----+---  +  -+ -- -- -- -- -- -- --
     1)     18921.2  Rx         0001  8  01 00 00 00 00 00 00 00
     2)     18922.3  Rx         0000  8  31 34 53 65 72 76 6F 4D
     3)     18922.7  Rx         0000  8  31 32 53 65 72 76 6F 4D
     4)     18926.4  Rx         0000  8  31 35 54 6F 6F 6C 48 4D
     5)     18927.9  Rx         0000  8  31 33 53 65 72 76 6F 4D
     6)    100904.4  Rx         0005  8  02 00 00 00 00 00 00 00
     7)    100904.4  Rx         0004  8  02 00 00 00 00 00 00 00
     8)    100904.4  Rx         0002  8  02 00 00 00 00 00 00 00
     9)    100904.4  Rx         0003  8  02 00 00 00 00 00 00 00
    10)    100905.8  Rx         0000  8  32 34 4C 69 6E 72 59 41
    11)    100906.6  Rx         0000  8  32 32 4C 69 6E 72 58 41
    12)    100909.7  Rx         0000  8  32 35 54 6F 6F 6C 44 52
    13)    100911.3  Rx         0000  8  32 33 4C 69 6E 72 5A 41
```

Figure 99: Screenshot of Data from CAN bus Monitor

The screenshot in Figure 99 is captured from the data dump from PCAN view, and shows: the number of the message, the timestamp of the message, the ID of the transmitting node and the data bytes included in the messages. Decoding of the message data bytes is done as per the message packet formatting presented in Section 6.7. The following information is used to analyze the response times:

- ID0x05: Fez Panda 2 board (Tool Module running at 72 MHz);
- ID 0x04: chipKIT board (Linear X Axis Module running at 80 MHz);
- ID 0x03: Fez Panda 2 board (Rotary A Axis Module running at 72 MHz);
- ID 0x02: Arduino UNO board (Linear Z Axis Module running at 16 MHz);
- Label A shows the general call made to all modules. The general call is a bus scan, to detect which modules are connected to the bus. The modules then respond with their unique ID to indicate their connection on the bus;
- Label B shows the transmitted data from the PC, requesting a data download from each distributed module;
- Label C shows the responses to the data download messages from each of the distributed modules.

Table 20 presents the response times computed from Figure 99. The following can be deduced from the information:

- Despite having the second fastest clock frequency, the Fez Panda 2 boards have the slowest response times;
- The Arduino UNO running at a clock frequency of 16 MHz, less than a quarter of the FEZ Panda 2 frequency, has a significantly faster response time;
- Despite the Arduino UNO running at a fifth of the clock frequency of the CHIPkit, it matches the response times;

- The Arduino UNO and the CHIPkit use similar Arduino bootloaders, unlike the FEZ Panda 2 board which runs on the Microsoft .NetMicro Framework. The response times are therefore not purely dependent on the microcontroller operating frequency, but are also based on the microcontroller architecture and microcontroller bootloader framework.

Table 20: Summary of Distributed Module Response Times

Message	**Response Time (ms)**			
Board	ID 0x05	ID 0x04	ID 0x03	ID 0x02
General Call	5.2	1.1	6.7	1.5
Data Download	5.3	1.4	6.9	2.2

9.9. *Chapter Summary*

Chapter 9 has presented the user with the test system used for testing the performance of the OACS on the MRMT. The test results for: accuracy, repeatability, response times, power usage, CPU load, algorithm outputs and reconfiguration times have all been shown and analysed. Following from these results, a discussion is included in Chapter 10 where the value add, successes and problems with the OACS is evaluated.

10. Discussion

The researched, designed and implemented OACS aims to verify whether the concept of modular machines and OA platforms can assist RMS to cope with the dynamic manufacturing environment. The MRMT had been developed previously, using mechanical modules. However the fixed electronic hardware and control software solutions implemented on the MRMT limited its reconfigurability. The discussion focuses on the test results covered in Chapter 9. Furthermore a discussion also follows evaluating the successes and limitations of the developed OACS and OA systems in general.

10.1 Performance of Electronic Subsystems

The performance of the OACS on the existing MRMT based on the results presented in Chapter 9 is discussed and evaluated in terms of accuracy, repeatability, power usage and vibration monitoring for machining and control.

10.1.1 Position Control: Accuracy and Repeatability

The performance of the system in terms of position accuracy and repeatability was evaluated based on a set of experimental tests performed on the system. All of the axes showed similar trends with similar encoder counts for CW and CCW rotations. The X axis and the CCW rotation for the Z axis met the desired repeatability requirements with worst case variations below 1%. On the other hand the CW rotation of the Z axis had a worst case variation of 1.97%. In terms of accuracy, 9 of 10 test runs on the translational axes have met the specification of accuracy to within 1 mm of the target point. Both translational axes produced 169 counts per mm which corresponds to the specified control resolution of $5.9x10^{-3}$ mm.

The Z axis has added weight due to the attached accessory and spindle modules. CW motion on the Z Axis corresponds to an upwards movement. Padayachee noted that the Z axis was designed with a low cost sliding mechanism and had the lowest stiffness of all the modules [29]. These added weights as well as the mechanical limitations on the module are the contributing factors that have ultimately affected the performance in terms of repeatability and accuracy as highlighted above.

The A axis failed to meet the accuracy specification, and further analysis revealed that the distributed module for the A axis, based on the FEZ Panda 2 microcontroller, failed to record all of the encoder counts. Based on the minimum control resolution, the expected count per degree for the rotary axes is 67 counts per degree. However the microcontroller only recorded 63 counts per degree. Further tests were conducted after lowering the encoder detection to 512 counts per rotation, and these tests showed worst case variation of 0.41% and accuracy of 1.45^{o}, which is acceptable as per the design specification. However, halving encoder resolution due to microcontroller performance is a trade-off as the control resolution is doubled from 0.015^{o} to 0.03^{o}. Despite the successes of this in terms of acceptable performance, it introduces a key challenge to performance of OA systems and multiple microcontroller implementations and this has been discussed in Section 10.6.

10.1.2. Axes Speeds and Time to Execute Commands

The time taken to execute the motion commands from the PC was captured, and from this it was evident that there was a time difference between CW and CCW motion for all axes. There was consistency with all the axes where the time to complete a command for CW motion was greater than the time for CCW motion. The time taken for CCW motion was on average 81.3% of the time taken for CW motion.

The 12 V DC motors used on the MRMT were slow to respond to acceleration commands. As a result, the acceleration component of the acceleration and deceleration routine was negated. The motors were driven to maximum speed and then the speed was reduced to allow the motors to reach their target speeds. This consistency in different times due to CW and CCW rotation of the electrical motors indicates that there is a loss of power and speed when the motors are driven in a counter clockwise direction. Based on the time observation, it is evident that the motors used, exhibited lower performance for CCW rotation as opposed to CW rotation and this was a contributing factor affecting overall system performance.

Moreover, the modules were required to meet minimum speed requirements as per design. The translational axes X and Z both met the minimum requirement of 100 mm/min. The X axis achieved a speed of 159 mm/min during CCW rotation and a speed of 130 mm/min for CW rotation. Similarly, the Z axis achieved a speed of 135 mm/min during CCW rotation and a speed of 110 mm/min for CW rotation. The rotary axes achieved a speed of 388 ^{0}/min for CCW rotation which met the minimum requirement of 360 ^{0}/min. However for CW rotation of the A axis, the maximum speed achieved was 314 ^{0}/min which was below the minimum requirement. It is likely that the lower speed performance of the A axis is due to the power transfer loss due to the efficiency rates of worm gear box systems [64].

10.1.3 Power Usage

The CCW average is greater than the CW average for X and A axes, where the CCW averages are 2.28 A and 1.97 A respectively. In addition to this, there is consistency between the X and A axes when the time to complete a command is compared to the current consumption. This comparison reveals that the CCW current consumption, which is greater than CW consumption, conforms to the shorter time.

For the Z Axis, which is the upright column, the CW average is greater than the CCW average current where the CW average is 4 A. CW motion on the Z Axis corresponds to an upwards movement, and as mentioned previously, the Z axis has additional weight load due to the attached accessory and spindle modules. Consequently a higher current consumption for upward motion as compared to downward motion is expected despite the shorter time. These are additional factors which have affected the performance of the Z axis in terms of accuracy, speed, repeatability and power usage.

The maximum current consumption of the axes, which occurs during starting motion of the motors, totals 15.3 A. Similarly the maximum current required for drill operation was 4 A. Based on these maximum currents, the existing 12 V 29 A power supply is sufficient for the system. Assuming similar current consumption for additional modules, the existing power supply is only able to supply 2 additional modules, after which it will need to be upgraded.

An analysis of the drill current consumption shows the current usage between 1.3 A and 4 A for drilling 40 mm into a product. As the drill depth increases, the current required also increases. Furthermore when the current consumption is compared to the vibrations recorded during drilling, there is a trend between the drill depth and vibration intensity. As the drill depth increases there are greater vibrations, therefore greater resistance and thus the current required increases.

Thus, if the MRMT is required to drill deeper, the current consumption would rise which could trip the power supply network. Therefore, running the system with these additional modules would not be advised as the system would be operating at its limit. An additional PSU can be added in series with the existing PSU in order to the increase the capacity of the power distribution network. This can be achieved with minimal interference to the existing system design which is part of the scalable features of the OACS. However, the idea of installing a system with excess capacity was one of the key problems with FMS due to the increased initial capital investment and therefore an alternate solution needs to be researched.

10.1.4 Vibrations

The current consumption for the translational axes shows a distinct sinusoidal like trend with the graph, showing several peaks. The number of peaks in the trend actually relates to the number of power screw revolutions that were needed for the commanded movement. This clearly indicated the added load on the PSU during a certain section of the rotation of the power screw. Coupled with the friction induced from movement due to mechanical imperfections, movement of the mechanical modules produced vibrations.

There was a distinct increase in recorded vibrations on the drill module from steady state drill operation to drilling into a product. Based on the accelerometer readings, it is seen that vibrations occur mainly from drill operation, but also from axial movement. The axial movement vibrations during steady operation result from uneven friction characteristics in the module drive and sliding systems [29], thereby confirming that the source of the vibrations is from the mechanical system and not induced by the control system. Furthermore as drill depth increases, comparing between 20 mm and 40 mm depth, the vibration intensity increased significantly. These increased vibrations add an additional load to the drill and Z axis, and this is seen by the increased current consumption during drilling operation.

10.1.5 Distributed Module Response Times

The response times of the distributed modules were analysed to determine effects of the multiple microcontroller implementation for the distributed modules. The Arduino UNO and the CHIPkit use similar bootloaders unlike the FEZ Panda 2 board which runs on the Microsoft .NetMicro Framework, which is slower than the Arduino based boards. Despite the Arduino UNO running at a fifth of the clock frequency of the CHIPkit, it closely matched the response times of the other boards, implying that at clock frequency of 16 MHz on the Arduino UNO is sufficient for operation of: the microcontrollers, communications network and host PC.

The minimum operating frequency and the corresponding response time is thus dependant on when the distributed module response time exceeds that of the sample time between PID controller iterations. In this OACS design, the PID sample time is set to 500 ms. Experimental tests revealed that when the FEZ Panda 2 Board clock frequency was lowered to 5 MHz, the response time exceeded the 500 ms limit set by the sampling time. This would then be the low-end cut off frequency after which the controller would fail.

However, the FEZ Panda 2 board, despite operating at 72 MHz, was unable to detect all of the encoder pulses from the rotary axis. This implies that despite the clock frequency, the performance of the distributed module microcontroller is also dependant on the bootloader.

In summary, the analysis revealed that the response times are not purely dependent on the microcontroller operating frequency, but are also based on the microcontroller architecture and microcontroller bootloader framework which poses a problem for OA systems.

10.2 Performance of OACS on PC

Further to the above control performance evaluation, the OACS is also evaluated on the CPU load, reconfiguration times and response times based on a multiple microcontroller implementation to evaluate the success of the developed system as a whole.

10.2.1 Reconfiguration Times

The introduction of OACS on an MRMT is aimed at assisting the MRMT to reduce the system downtime during reconfigurations. Therefore the time that the OACS takes to reconfigure itself due to the changes in the mechanical modules, is a critical assessment point of OACS. Experimental tests were conducted to get reconfiguration times from users familiar to the OACS and from new users as well.

From test data, the reconfiguration and setup time of the OACS with 3 modules was, on average, 200 seconds when conducted by a familiar user and 366 seconds by a new user. Similarly, for a system with 4 modules, the time taken on average for configuration by a familiar user was 240 seconds and 440 seconds for a new user. The test data shows that on average it takes a new user approximately 1.75 times longer (than a familiar user) to reconfigure the OACS.

Although these results are “soft results” with no real comparison, it is evident that due to the modular nature of the OACS, as well as the seamless integration due to the plug and play features which automatically determine active modules and download information from these modules for configuration and control allow for the OACS to be configured for operation within minutes. Therefore, the easy software reconfiguration assists the MRMT in reducing the down time between machine changes.

10.2.2 CPU Load

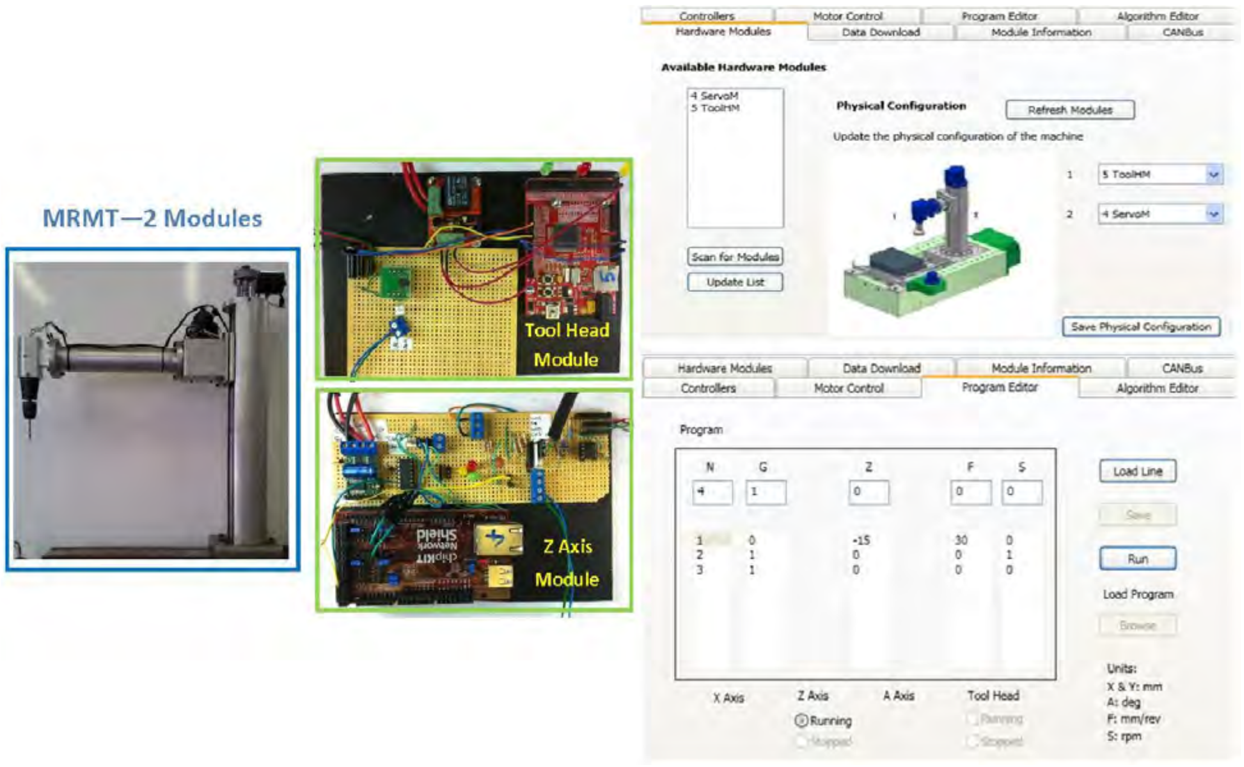
The load on the CPU was monitored and on average the CPU usage is less than 4%. As expected during operation and configuration the load increases and peaks at 38%. This peak is during machining command calculations as the interpolation, acceleration and deceleration and the control routines are run. Although these results are also “soft results” with no direct comparison, the load on the CPU shows that the OACS operating for a 4 axis system does not overload a standard Windows PC.

10.3 OACS Integration with Mechanical System

Figure 100 and Figure 101 illustrate how the OACS integrates with the mechanical system based on varying mechanical system configurations and DOF, which then translates to different controller configurations. The following images illustrate how the OACS can dynamically reconfigure itself based on the available hardware modules that are active on the bus, and how the OACS limits access to the user depending on what physical configuration is entered by the user.

Figure 100 shows the MRMT configured with 2 modules, the Tool module and Z axis, whereas Figure 101 shows a 4 module configuration. Although only screenshots of the Physical Configuration and the Program Editor Windows are shown, the other parameters and aspects of the OACS are also automatically configured based on the mechanical system configuration.

Although not visible to the user, as discussed in Section 8.4, the OACS temporarily stores the downloaded data from the distributed modules in temporary module classes derived from a generic class. Thereafter, when the user enters the machine configuration, a new class is derived from generic tool, rotary axis or translational axis classes, and the data from the temporary class is copied to the new derived class. This step is critical in evaluating what functions are necessary for the OACS and which display options are to be configured for the GUI as shown in Figure 100 and Figure 101.



4 ServoM
5 ToolHM

Update the physical configuration of the machine

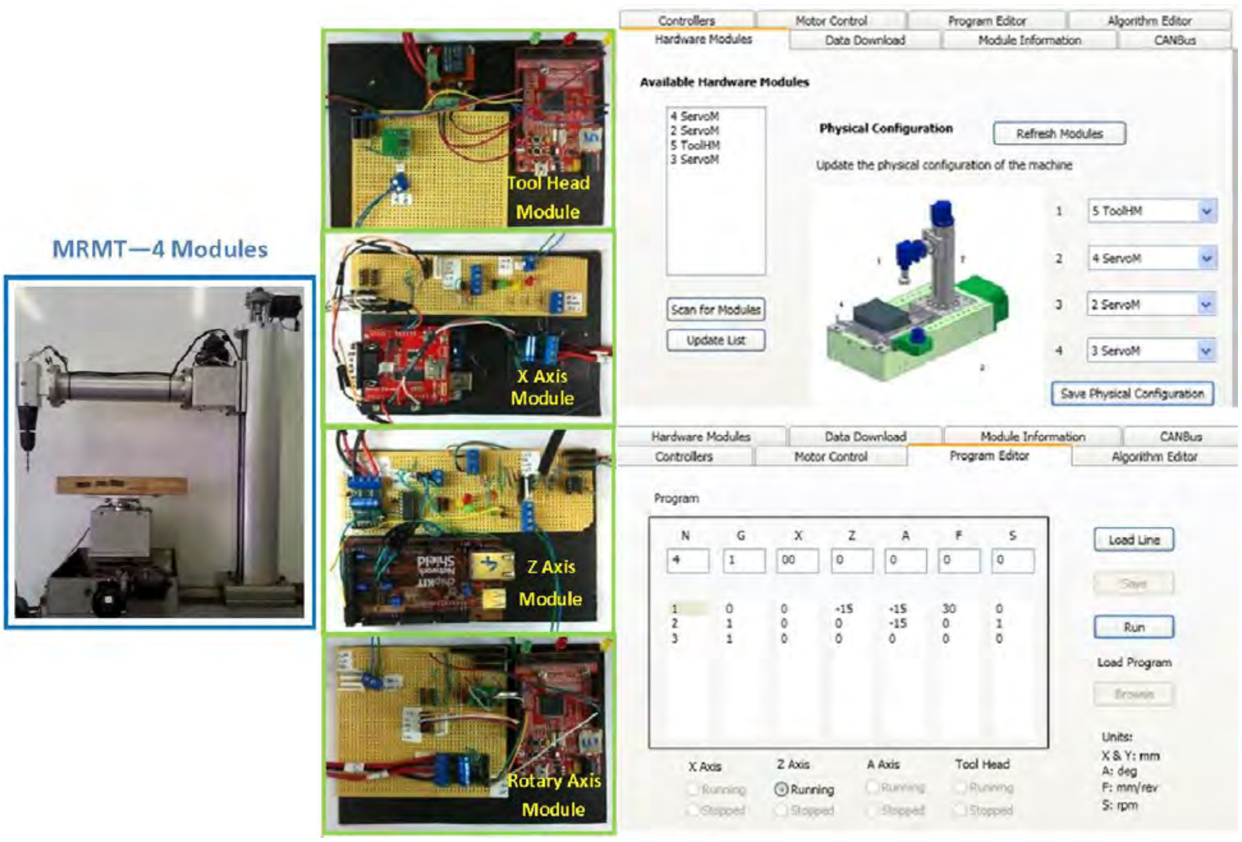
1 5 ToolHM
2 4 ServoM

Program

N	G	Z	F	S
4	1	0	0	0
1	0	-15	30	0
2	1	0	0	1
3	1	0	0	0

Units:
X & Y: mm
A: deg
F: mm/rev
S: rpm

Figure 100: MRMT and OACS Configuration 2 Modules



Available Hardware Modules

4 ServoM
2 ServoM
5 ToolHM
3 ServoM

Physical Configuration

Update the physical configuration of the machine

1	5 ToolHM
2	4 ServoM
3	2 ServoM
4	3 ServoM

Program

N	G	X	Z	A	F	S
4	1	00	0	0	0	0
1	0	0	-15	-15	30	0
2	1	0	0	-15	0	1
3	1	0	0	0	0	0

X Axis	Z Axis	A Axis	Tool Head
Running	[✓ Running]	Running	Running
Stopped	Stopped	Stopped	Stopped

Load Program

Units:
X & Y: mm
A: deg
F: mm/rev
S: rpm

Figure 101: MRMT and OACS Configuration 4 Modules

10.4 OACS and OA standards

The aim of an OACS is to verify whether the concept of modular machines and OA platforms can assist RMS to cope with the dynamic manufacturing environment. Therefore the OACS was designed with a set of core principles that are common to RMS and OA standards with the aim of proving that OACS are vital for the realization of MRMT and RMS in general. Table 21 summarises the core principles and describes how the various aspects of the research and design have incorporated and displayed these principles. Since the control system was located on the MRMT, similar core concepts such as: reconfigurability, modularity and scalability were required for the control system.

Table 21: Summary of OACS Core Principles

	Electronics Subsystems	**PC based OACS**	**Communications Interface**
Core Principle			
Modularity	Stand-alone distributed modules that are customized to interface the necessary sensors and customized circuitry	Software modules create one to one link between hardware	2 of the microcontrollers have plug and play boards that provide CAN communication with the third requiring design of an external CAN transceiver circuit
Scalability	Scalability is achieved through modularity and the supporting communications	Object orientated design that derives live time classes from generic classes for the number of modules	Plug and play communications network that allows up to 127 nodes to be added

	interface	connected to the system bus	
Interoperability	Multiple microcontroller implementation	Generic Windows OACS application development to run on a standard Windows PC	Any distributed module microcontroller with either built in CAN module or CAN bus circuitry can be added to the bus
User Customization	Buffer layers allow any sensor/microcontroller/axis to be added to the system as long as the data exchange standard between layers is adhered to	User specific programs can be programmed into the system. Users can choose controllers based on preference. Users can edit control algorithms	End user to defines the data exchange and packet messages
Vendor Neutrality	Multiple microcontrollers and integration of off the shelf sensors with customization on microcontrollers	Generic Windows OACS application development to run on a standard Windows PC	CAN bus is an open access communication protocol allowing the end user to define the data exchanges

Furthermore, the existing industrially available control solutions were closed systems and therefore were only allowed reconfigurability and scalability based on the vendor's products and add-on modules. The discussion in Chapter 3 presents a selection of industrially available control solutions. It also highlights the academic research attempts at developing an OA system. Based on the aforementioned research, it was found that design ideas such as: embedded distributed modules, a plug and play interface and a one-to-one software and hardware mapping, are common to many of these OA systems. The OACS was designed and developed according to these principles. Table 22 presents a comparative analysis between the OACS and the solutions presented in Chapter 3.

Table 22: Comparison of OACS and Other Research Attempts

	OACS	**DeltaV**	**Siemens**
Electronic System	Distributed modules	DCS	PLC with modular add-ons
Software System	Open VN system	VS system	VS system
Flexibility	Yes	I/O only	Modular add-ons
Reconfigurability	Hardware and software	I/O only	Yes
Customization	Hardware and Software	Limited in software	Limited in software
Vendor Neutrality	Yes	No	Limited Add-on modules
Mechanical Platform	Hybrid MRMT	N/A	Customizable
Manufacturing Performance		N/A	PLC optimised for fast accurate machine control
Software Scalability	Yes	No	No
Hardware Scalability	Yes	I/O only	Yes

The core principles of the OACS, together with the comparative analysis, shows how the OACS moves towards a VN solution, embedded with the core ideas of RMS, to assist RMT in achieving reconfigurability in hardware and software. Furthermore, the research has shown that design ideas such as: embedded distributed modules, a plug and play interface and a one to one software and hardware mapping, can all be successfully implemented on a VN system. The VN system goes further to allow third party add-ons and COTS modules or sensors with customization to the hardware or software on the distributed module, thereby maximising the possibilities for reconfiguration.

The OACS differs from other research attempts in that it aims at developing an open system from the base up through its multiple microcontroller implementations. Furthermore, the system shows its customizability and flexibility by allowing the integration of COTS modules and any sensor module to any distributed module, with invisibility through a well-defined interface between tiers of the system architecture.

10.5 OACS on MRM for RMS

The idea of a multiple microcontroller implementation allows end user customization, and illustrates that the system is not limited in its software and electronic architectures. Testing has revealed that with three different microcontrollers, the system exhibited acceptable performance results, but also revealed key areas that could be problematic for the realisation of OA systems. The distributed modules also divide the system architecture into different layers thereby creating buffer layers as discussed in Chapter 5.

These two layers create a buffer layer or an invisible layer between each of the three tiers. This buffer layer allows for the components at each tier to be of many variations, types and architectures. Consequently all that is required to ensure system functionality is that the interface and data transfer between the tiers is consistent and standardised.

The functioning of the OACS is therefore independent of the types of modules at each tier, or what is located at each tier. The components of a module in a tier can be modified, changed or upgraded without affecting system functionality, thus making the system flexible and reconfigurable. Furthermore the OACS allows seamless integration of new modules or sensors via scalable, robust and flexible plug and play communications interface between layers.

In the case of an axis or servo or sensor module being added or reconfigured, the distributed module microcontroller does not need to be changed, but rather needs to allow for software to be reprogrammed or updated, thereby decreasing the cost of upgrades or modifications to the system. Similarly, COTS components can be integrated onto the MRMT, regardless of the type and functioning of the component added. All that is required to ensure system functionality is customization on the distributed module to ensure that the correct data exchange protocols are adhered to.

For example, if the added component is a motor driver that uses a digitally encoded communication as opposed to PWM signals, the distributed module software routine can be customized to cater for this. Similarly, the interfacing hardware on the distributed module can be customised to ensure that the system maintains the necessary functionality.

The OACS also allows algorithms to be modified by the end user. The routines can be edited and adapted by ensuring that the pointers to the input data and output data adhere to the original routine. Once a new routine is added to the OACS, the system will need to be recompiled. Changing this routine does not affect the performance and operation of the rest of the OACS, making the OACS reconfigurable and customizable as per the user requirements. The OACS exhibits its flexibility and customizability by allowing users to customize the control algorithms and to incorporate user specific programs. The OACS has been developed with PID control algorithms, although if desired, this control routine can be: edited, the system recompiled and performance re-evaluated.

The flexibility, reconfigurability and modular structure allows the system to be gradually upgraded over time as production requirements change. This therefore allows the system to be installed with the functions required, at a given time, thereby minimizing the initial capital investment required, which was a key problem of FMS. Customization allows newer modules to be integrated, and the flexible and adaptable software and control means that the machine can modify its functionality thereby also prolonging its lifecycle. In a plant environment, modules can also be re-used and interchanged between different machines as per production requirements. This further decreases the costs for changes when the production requirements change.

Furthermore, as the lifecycle of the MRMT increases, users may upgrade and optimize modules on the MRMT by replacing a component and by customizing the respective distributed module. Similarly, the system can be installed and designed with cheaper/lower performance sensors or modules to lower the initial capital investment. Additionally, as production requirements change or improved performance is required, only the low performance sensors or modules need to be changed.

For example, on the existing MRMT that the OACS was developed for, the system uses 12V wiper motors which exhibit low torque and the inability to operate at low voltages. These motors can be upgraded, and with customization to the motor driver circuitry and distributed module software, the OACS can have improved performance.

Furthermore, the OACS has proven that an open system can allow end user customization throughout the life time of the MRMT by allowing access to change and modify performance parameters such as controller gains. Similarly, this can be up scaled to include additional end user customization and reconfiguration options such as allowing users to change the interpolation times, cycle refresh times or even PID sample times.

It is now evident that due to the flexible, reconfigurable and modular nature of the OACS in its electronic, control and software architectures, the OACS allows MRMT to help RMS by:

- Reducing the time for reconfiguration;
- Assisting in the rapid introduction of new technologies or modules;
- Reducing the cost of system upgrades.

10.6 Challenges and Limitations for OA Systems

The performance results from the multiple microcontroller implementations of the distributed modules on the OACS showed one of the areas of concern for OA systems. The requirement for: end user customization, VN and integration of off-the-shelf components led to the idea of a multiple microcontroller implementation. However the system implementation has shown that if the system is open and not defined within guidelines, it may ultimately have a negative impact on the system performance.

Furthermore the three microcontrollers chosen to implement the distributed modules varied in: architecture, bootloaders and clock frequency. The operating frequencies of the distributed modules and the microcontroller bootloaders pose an additional challenge to the: operation, performance and integration of the system. The developed OACS showed that despite the differences in frequencies and bootloaders, the distributed modules all managed to communicate with the host PC. However the problems occurred with the reading of the encoder from one of the distributed modules.

The MRMT will be performing CNC machining tasks in the manufacturing environment; therefore accuracy and repeatability are critical points of success. Thus, performance trade-offs as demonstrated by this implementation will not be acceptable for a commercially viable solution.

The problems and findings of this research have shown that unless unified standards and guidelines for OA systems are developed and adhered to in system design, OA systems will be faced with numerous challenges that will hinder and limit the machine performance. These guidelines need to cover the standards for data exchange between the tiers in the architecture, as well as the types of microcontrollers in terms of bootloaders and operating frequencies which have been tried, tested and proven.

The OACS allows users to edit control algorithms provided that the input and output parameters are matched to the existing data pointers. Although when an algorithm is edited, the system needs to be recompiled. The process of recompiling the program, after an algorithm is edited, is functional though not ideal. In addition, a further aim of OA platforms is to allow 3rd party add-ons and external interfaces for programs and algorithms, although this falls out of the scope of this research. Further research should then investigate how the new algorithms can be incorporated during system runtime.

The flexibility of the OACS, due to the multi-tier architecture, which creates buffer layers between the tiers, can allow integration of COTS modules and different sensors to each distributed module. Since the OACS uses the lowest common denominator in terms of feed rates, to ensure complete system functionality, the integration of COTS modules or sensors can have a negative effect on system performance. This intensifies the need for unified standards and guidelines to OA system development as well as the need for further research into the development and integration of COTS modules onto an existing MRMT.

This OACS was developed and tested for the hybrid MRMT, which has the motor for each module located on each module to allow for modularity and easy system reconfiguration. This mechanical design leads to an unfavourable mass distribution on the machine due to deflections [29]. Furthermore, the Z axis has additional mass due to the attached accessory module and process module which has a negative effect on performance as discussed. The performance of OACS is therefore also dependant on the mechanical integrity of the mechanical systems. Further research should look at the possibility of redesigning the mechanical platform with greater rigidity, frictionless contacts and a more favourable mass distribution to improve system machining performance.

Although this would have a cost impact, the current mechanical system design was built as a low cost prototype and therefore not the ideal test bench for the OACS [29].

The number of active modules on OACS and the MRMT has an impact on the load of the power supply system. The system can undergo a reconfiguration where new modules can be added, existing modules can be upgraded, or newer and better motors can replace the existing motors. Under such conditions, the overall load on the power supply system can increase due to more powerful motors or additional sensors. The current OACS design can cater for 2 additional mechanical modules and its associated distributed module and sensors. However, this solution to install a system with excess capacity, like FMS, is not cost effective. Thus further research needs to be conducted to evaluate the feasibility and practicality of a scalable power supply system. A fixed power supply system on the MRMT is currently the only component that remains fixed and not scalable, and further research aligned with the aims of modularity and scalability will benefit MRMT and RMS. The inclusion of such a modular and scalable design on MRMT systems can also contribute to the cost effectiveness and reconfigurability of MRMT and RMS in general.

The aforementioned points have indicated that due to the openness of the OA system and the need for: end user customisation, VN and flexibility, the OA system requires well defined standards and guidelines for development. OA systems in mobile phone industry have suffered as a result of a lack of a unified standard among a vast number of platforms that software systems were being developed for [65]. Furthermore, it was only after Android was developed for mobile platforms that there was cohesion and alignment in development that allowed for rapid innovation and development of open mobile system development [65]. Similarly, the OA systems for machine tools require standards and guidelines, and in the long term, if these are not developed for OA systems, the performance of OA systems on MRMT for RMS can be hindered.

Furthermore, the mechanical integrity of the MRMT of mechanical system that the OA system is developed for, also plays a critical impact on the performance of the complete system.

10.7 Chapter Summary

Chapter 10 has presented a discussion on the performance of the researched, designed and developed OACS which was tested on an existing MRMT. The discussion highlights the key successes and achievements of the OACS. It also indicates problematic performance areas and goes on to evaluate the reasons for these failures. Furthermore, the key characteristics of the OACS are compared to the key characteristics of OA systems, and a discussion has covered and evaluated the successes of the OACS based on the aims and objectives of this research. Finally, a summary of the challenges and limitations facing OACS and OA systems in general has been presented alongside key areas for future research.

11. Conclusion

The present day manufacturing environment has forced manufacturing systems to be flexible and adaptable in order to a match the product demands and frequent introduction of new products. Based on the literature review presented, which covers the current day manufacturing paradigm, it has been shown that the reconfigurable manufacturing paradigm looks to incorporate the advantages of DMS and FMS, to create system that allows high throughput and flexibility over time. Past research has further shown that the lack of development of RMT and OA platforms is currently the limiting factor to the establishment of RMS. These factors have been the primary motivation for this research.

The research has proposed, designed and developed a novel solution that incorporates the core principles of RMS and OA platforms. The OACS has been developed as a modular solution that links closely to the existing modularity on the RMT. The aim was to create a one to one link between mechanical and electronic hardware and the software system. This has been achieved by the addition of embedded microcontroller based distributed modules, which interface the electro-mechanical machine axes as well as the host PC, via the CAN bus communication interface.

The OACS is dynamic in its setup as is able to conduct live time scans to determine what modules are active on the machine tool. On the host PC, the user is presented with a GUI which allows the user to configure the control system based on the MRMT physical configuration and the DOF available. The physical machine configuration entered is used to determine the kinematic viability of the machine. Based on the current configuration, the system automatically adapts the GUI to allow the user access to functions relevant to the machine structure. The underlying software algorithms such as: text interpretation, linear interpolation, PID/PI controllers and kinematic viability, are all part of the OACS and are used at run time for machine operation.

For the successful one to one implementation, the system uses distributed modules that interface the mechanical modules and the host PC. The distributed modules are based on different microcontrollers which have successfully demonstrated the openness and customizability of the system. The multiple microcontroller design allows end user customization and illustrates that the system is not limited in its software and electronic architectures.

The OACS allows end user customization in: software, electronic hardware and mechanical hardware. Software customization is achieved through: customization of control algorithms, electronic hardware customization through the open and multiple microcontrollers and mechanical hardware customization through selection of different library modules for machine configuration. Furthermore, the distributed modules allow customisation in terms of COTS sensors or actuators that can be integrated with any distributed module by customization on the module.

The interface to the host PC is designed with CAN bus communication network which supports extendibility, interoperability and scalability. These features assist the whole system to be modular, scalable and reconfigurable by allowing: seamless integration, and addition or removal of electro-mechanical modules through a well-defined plug and play interface.

The discussion has shown how the OACS moves towards a VN solution embedded with the core ideas of RMS, to assist RMT in achieving reconfigurability in hardware and software. Furthermore, the research has shown that design ideas such as: embedded distributed modules, a plug and play interface and a one to one software and hardware mapping, can be successfully implemented on a VN system.

The flexibility of the OACS due to the multi-tier architecture which creates buffer layers between the tiers allows integration of COTS modules and different sensors or actuators to each of the distributed modules furthering the customizability of the system. The functioning of the OACS is also independent of the types of modules at each tier or what is located at each tier provided the data exchange between tiers in constant. The components of a module in a tier can therefore change and be upgraded without affecting system functionality thus making the system flexible and reconfigurable.

Future research should look at the development of guidelines and standards for data exchange between the tiers in the architecture, as well as the types of microcontrollers in terms of bootloaders and operating frequencies to avoid control performance limitations. Additionally, the OACS needs to be recompiled when algorithms are edited, although functional it is not ideal. Thus the possibility for on-line recompilation and for the integration of third party algorithms should be investigated. The performance of OACS is dependent on the mechanical integrity of the mechanical systems. Further research should look at the possibility of redesigning the mechanical platform with greater rigidity, frictionless contacts and a more favourable mass distribution than the MRMT to improve system machining performance. Finally, the number of active modules on OACS and the MRMT has an impact on the load of the power supply system. Further research needs to be conducted to evaluate the feasibility and practicality of a scalable power supply system as setting up a system with excess capacity has cost implications and was one of the drawbacks of FMS. The inclusion of such modular and scalable designs on MRMT systems will contribute to the cost effectiveness and reconfigurability of MRMT and RMS in general.

The flexibility, reconfigurability and modular structure allows the system to be gradually upgraded over time as production requirements change. This then allows the system to be installed with the functions required, at a given time, thereby minimizing the initial capital investment required.

Furthermore, as the lifecycle of the MRMT increases, users may upgrade and optimize modules on the MRMT by replacing that component. By customizing the distributed module, the system can then ensure correct functionality. The research has shown that the flexible, reconfigurable and modular nature of the system in its electronic, control and software architectures assists the system to satisfy the aims of OACS. Furthermore the distributed electronic modules, the intelligent communications network and modular software system, which form the basis of the OACS, assist RMT's and RMS to:

- Have greater flexibility;
- Assist in the rapid introduction of new technologies or modules;
- Reduce system upgrade costs;
- Provide consistent and repeatable control;
- Prolong the beneficial operation period of the MRMT.

References

1. Setchi, R.M. and Lagos, N. *Reconfigurability and Reconfigurable Manufacturing Systems: State-of-the-Art Review*. in *2nd IEEE International Conference on Industrial Informatics*. 2004. Berlin Germany: IEEE.
2. Bi, Z.M., Lang, S.Y.T., Verner, M., and Orban, P., *Development of Reconfigurable Machines.* International Journal of Advanced Manufacturing Technology, 2008. 39(11): p. 1227-1251.
3. Mehrabi, M.G., Ulsoy, A.G., Koren, Y., and Heytler, P., *Trends and Perspectives in Flexible and Reconfigurable Manufacturing Systems.* Journal of Intelligent Manufacturing, 2002. 13(2): p. 135-146.
4. ElMaraghy, H.A., *Flexible and Reconfigurable Manufacturing Systems Paradigms.* International Journal of Flexible Manufacturing Systems, 2005. 17(4): p. 261-276.
5. Katz, R., *Design Principles of Reconfigurable Machines.* The International Journal of Advanced Manufacturing Technology, 2007. 34(5-6): p. 430-439.
6. Padayachee, J., Bright, G., and Masekamela, I., *Modular Reconfigurable Machine Tools: Design, Control and Evaluation.* South African Journal of Industrial Engineering, 2009. 20(2): p. 127-143.
7. Landers, R.G., Min, B.K., and Koren, Y., *Reconfigurable Machine Tools.* CIRP Annals - Manufacturing Technology, 2007. 50(1): p. 269-274
8. Jimenez, C.H.O. and Salinas, I.E. *A Glance at Reconfigurable Manufacturing Systems (RMS): Possible Connotation on a Path To High Performance*. in *6th Latin American and Caribbean Conference for Engineering and Technology*. 2008. Tegucigalpa, Honduras: LACCEI.
9. Setlak, G. and Pieczonka, S., *Design Concept of Intelligent Management Systems*, in *International Book Series Information Science and Computing*2009. p. 142.
10. Devedzic, V. and Radovic, D., *A framework for building intelligent manufacturing systems.* IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews, 1999. 29(3): p. 422-439.
11. Morales-Velazquez, L., Romero-Troncoso, R.d.J., Osornio-Rios, R.A., Herrerra-Ruiz, G., and Cabal-Yepez, E., *Open Architecture System Based on Reconfigurable Hardware-Software Multi Agent Platform for CNC Machines* Journal of Systems Architecture, 2010. 56(9): p. 407-418.
12. Koren, Y., Heisel, U., Javone, F., Moriwaki, T., Pritschow, G., Ulsoy, G., and Brussel, H.V., *Reconfigurable Manufacturing Systems.* CIRP Annals - Manufacturing Technology, 1999. 48(2): p. 527-540.
13. Stecke, K.E., *Flexibility is the Future of Reconfigurability. Paradigms of Manufacturing—A Panel Discussion*, in *3rd Confereence on Reconfigurable manufacturing*2005: Ann Arbour, Michigan, USA.
14. Koren, Y. *Reconfigurable Manufacturing and Beyond*. in *CIRP 3rd International conference on Reconfigurable Manufacturing*. 2005. Ann Arbor, Michigan, USA.
15. Koren, Y., *General RMS Characteristics. Comparison with Dedicated and Flexible Systems*, in *Reconfigurable Manufacturing Systems and Transformable Factories* A.I. Dashchenko, Editor 2006, Springer: Berlin. p. 27-45.
16. Maier, F. and Schroeder, R., *Competitve Product and Process Technology*, in *High Perfromance Manufacturing, Global Perspectives*, R.G. Schroeder and B.B. Flynn, Editors. 2001, John Wiley & Sons: New York. p. 93-114.
17. Mpofu, K., Kumile, C.M., and Tale, N.S. *Adaption of Commercial Off the Shelf Modules for Reconfigurable Machine Tool Design*. in *15th International Conference on Mechatronics and Machine Vision in Practice (M2VIP)*. 2008. Auckland, New Zealand: IEEE.
18. Pritschow, G., Altintas, Y., Jovane, F., Koren, Y., Mitsuishi, M., Takata, S., Brussel, H.v., Weck, M., and Yamazaki, K., *Open Controller Architecture – Past, Present and Future* CIRP Annals - Manufacturing Technology, 2001. 50(2): p. 463-470

19. Koren, Y., *Open-Architecture Controllers for Manufacturing Systems - Summary of Global Activity*, in *ITIA Series*1998. p. 15-37.
20. Oliveira, A., Pieri, E.D., and Moreno, U., *An Open-Architecture Robot Controller Applied to Interaction Tasks*, in *Advances in Robot Manipulators*2010, Ernest Hall.
21. Padayachee, J., Masekamela, I., Bright, G., Tlale, N., and Kumile, C., *Modular Reconfigurable Machines Incorporating Modular Open Architecture Control* in *15th International Conference on Mechatronics and Machine Vision in Practice (M2VIP'08)*2008: Auckland, New Zealand.
22. Birla, S., Faulkner, D., Michaloski, J.L., Sorenson, S., Weinert, G., and Yen, J.H. *Reconfigurable Machine Controllers using the OMAC API*. in *Proceedings of the CIRP 1st International Conference on Reconfigurable Manufacturing*. 2001. Ann Arbor, Michagan, USA.
23. Emerson-Process-Management, *DeltaV Digital Automation System System Overview*, 2009, Emerson: Austin, Texas.
24. Siemens-AG, *SIMATIC Controllers:The innovative solution for all automation tasks*, 2011: Germany.
25. Xiong-bo, M., Zhn-yu, H., Yong-zhang, W., and Hong-ya, F., *Development of a aPC-based Open Architecture Software-CNC System.* Chinese Journal of Aeronautics, 2007. 20(3): p. 272-281.
26. Farooq, M. and Wang, D.B., *A Reconfigurable and Modular Open Architecture Controller: The New Frontiers.* International Journal of Automation Technology, 2008. 2(3): p. 205-214.
27. Proctor, F.M., Shackleford, W., Yang, C., Barbera, T., Fitzgerald, M., Frampton, N., Bradford, K., Koogle, D., and Bankard, M. *Simulation and Implementation of an Open Architecture Controller*. in *Proceedings of the SPIE International Symposium on Intelligent Systems and Advanced Manufacturing*. 1995. Philadelphia, PA, USA.
28. Suh, S.-H., Kang, S.-K., Chung, D.-H., and Stroud, I., *Theory and Design of CNC Systems*. Springer Series in Advanced Manufacturing, ed. P.D.T. Pham2008, Verlag, London, UK: Springer. 466.
29. Padayachee, J., *Development of a Modular Reconfigurable Machine for Reconfigurable Manufacturing Systems*, in *School of Mechanical Engineering*2010, University of KwaZulu-Natal: Durban South Africa.
30. Tabbara, B. *Breathing Life Into Hardware and Software Codesign*. Embedded Systems Programming 2005 June 2011 [cited 18 4]; Available from: http://www.embedded.com/design/embedded/4006444/Breathing-life-into-hardware-and-software-codesign.
31. Isermann, R., *Modelling and Design Methodology for Mechatronic Systems.* IEEE/ASME Transactions on Mechatronics 1996. 1(1): p. 16-28.
32. Kommareddy, S., Kazuo, Y., and Yoshihito, K. *PC-Based Open Architecture Servo Controller For CNC Machining*. in *The Second Real Time Linux Workshop*. 2000. Orlando.
33. GHIelectronics *FEZ Panda || Board*. 2011.
34. Digilent *chipKIT Max32 Boar Reference Manual*. 2012.
35. Arduino. *Arduino Uno*. 2012 [cited 2012 February]; Available from: http://www.arduino.cc/en/Main/arduinoBoardUno.
36. Powell, J. *PROFIBUS PA and Foundation Fieldbus - a cost comparision*. 2007. 4.
37. Unknown. *Modbus FAQ*. 2011 [cited 2011 August]; Available from: http://www.simplymodbus.ca/faq.htm.
38. Unknown. *CAN - a brief tutorial*. 2011 [cited 2011 10 October]; Available from: http://www.computer-solutions.co.uk/info/Embedded_tutorials/can_tutorial.htm.
39. Nilsson, S. *Controller Area Network - CAN Information*. 2011 [cited 2011 14 October 2011]; Available from: http://hem.bredband.net/stafni/developer/frames.htm.
40. Unknown *The basics of Intrinsic Safety*. 1998. 4.

41. Unknown. *What is CAN Bus*. 2011 [cited 2011 9 October 2011]; Available from: http://www.canbuskit.com/what.php.
42. PEAK-System, *PCAN-PCI - PCI to CAN Interface - Uer Manual v2.0.*, 2011.
43. Texas-Instruments, *3.3-V CAN Transceivers*, 2011.
44. Digilent *chipKIT Network Shield Board Reference Manual*. 2012.
45. sparkfun-Electronics. *CAN-BUS Shield*. 2012 [cited 2012 Februarty]; CAN-BUS Shield for Arduino]. Available from: https://www.sparkfun.com/products/10039\.
46. Pololu. *Pololu High-Power Motor Driver 18v15*. 2011 [cited 2011 5 May 2011]; Available from: http://www.pololu.com/catalog/product/755.
47. Vorkoetter, S. *Electromagnetic Interference Reduction*. 2010 [cited 2011 10 October]; Available from: http://www.stefanv.com/rcstuff/qf200005.html.
48. Octopart. *Omron V-103-1A4*. 2012 [cited 2012 October]; Available from: http://sigma.octopart.com/8504533/image/Omron-V-103-1A4.jpg.
49. Issa, G. *Beginners guide to c# and .NET Micro Framework*. 2010.
50. Hewes, J. *The Electronics Club*. 555 and 556 Timer Circuits 2011 [cited 2011 10 October 2011]; Available from: http://www.kpsec.freeuk.com/555timer.htm.
51. Hewlett-Packard, *Two and Three Channel Optical Encoders*, 2011.
52. Analog-Devices *ADXL335*. 2009. 16.
53. Sparkfun-Electronics. *Triple Axis Accelerometer Breakout - ADXL335*. 2012 [cited 2011 September]; Available from: https://www.sparkfun.com/products/9269.
54. Tilbury, D. and Kota, S. *Integrated machine and control design for reconfigurable machine tools*. in *Proceedings of IEEE/ASME International Conference on Advanced Intelligent Mechatronics*. 1999. Atlanta, GA, USA: IEEE.
55. Hlavac, V. *Robot Kinematics*. 2011 September 2011; Pages 21]. Available from: http://cmp.felk.cvut.cz/~hlavac/TeachPresEn/51Robotics/11KinematicsRobot.pdf.
56. Unknown. *Forward Kinematics: The Denavit-Hartenberg Convention*. Chapter 3 p.71-102]. Available from: http://www.cs.duke.edu/brd/Teaching/Bio/asmb/current/Papers/chap3-forward-kinematics.pdf.
57. Murray, R.M., Li, Z., and Sastry, S.S. *A Mathematical Introduction to Robotic Manipulation*. 1994 November 2011]; Available from: http://www.cds.caltech.edu/~murray/mlswiki.
58. Franklin, G.F., Powell, J.D., and Emami-Naeini, A., *Feedback Control of Dynamic Systems*. Sixth Edition ed2006, Upper Saddle River, NJ, USA: Pearson.
59. Unknown. *DC Motor Speed: System Modeling*. 2012 [cited 2012 February]; Available from: http://ctms.engin.umich.edu/CTMS/index.php?example=MotorSpeed&section=SystemModeling.
60. Krass, M. *PID Control Theory*. 2006. 6.
61. Beauregard, B. *Improving the Beginner's PID – Introduction*. 2011 April 15th, 2011 [cited 2011 September 2011]; Available from: http://brettbeauregard.com/blog/2011/04/improving-the-beginners-pid-introduction/.
62. Zhong, J. *PID Controller Tuning: A Short Tutorial*. 2006. 13.
63. Smid, P., *CNC programming handbook: a comprehensive guide to practical CNC programming*2003: Industrial Press Inc.
64. Beardmore, R. *Gear Efficiency*. 2013 20/01/2013 [cited 2013 02/2013]; Available from: http://www.roymech.co.uk/Useful_Tables/Drive/Gear_Efficiency.html.
65. Vaughan-Nichols, S.J. *Andy Rubin, Android's founder, leaves project*. 2013 13 March 2013 [cited 2013 March 2013]; Available from: http://www.zdnet.com/andy-rubin-androids-founder-leaves-project-7000012563/?s_cid=e589.

Appendices

Appendix A: *Kinematic Modelling Data*

X Axis - Base Module

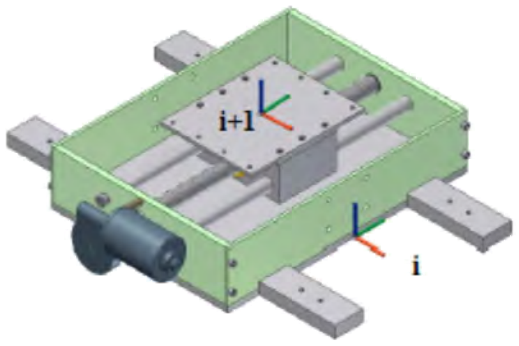


Figure A102: X Axis with i and i+1 Reference Points

Table A23: X Axis Design Data

Offset		Euler Angles	
X	Determinant	γ	0
Y	0	β	0
Z	-111	α	0
Range of Motion	[433 0 733]	Initial Position	583

$$ {}^{i+1}_{i}M_{X\ Axis} = \begin{bmatrix} 1 & 0 & 0 & x \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & -111 \\ 0 & 0 & 0 & 1 \end{bmatrix} $$

Z Axis - Column Module

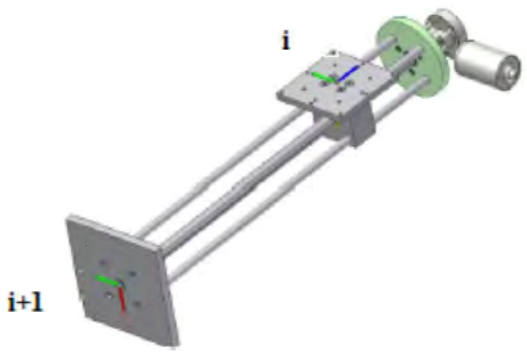


Figure A103:Z Axis with i and i+1 Reference Points

Table A24: Z Axis Design Data

Offset		Euler Angles	
X	-54.5	γ	0
Y	0	β	0
Z	Determinant	α	0
Range of Motion	[85 0 685]	Initial Position	685

$$^{i+1}_{\ \ i}M_{Z\ Axis} = \begin{bmatrix} 1 & 0 & 0 & -54.5 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

A Axis – Rotary Module

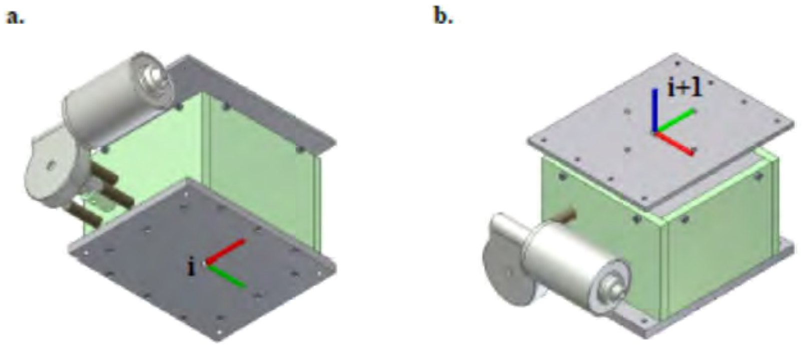


Figure A104: (a) A Axis with i (b) i+1 Reference Points

Table A25: A Axis Design Data

Offset		Euler Angles	
X	0	γ	0
Y	0	β	0
Z	-152	α	Determinant
Range of Motion	[-180 0 180]	Initial Position	0

$$^{i+1}_{\ \ i}M_{A\ Axis} = \begin{bmatrix} c\alpha & -s\alpha & 0 & 0 \\ s\alpha & c\alpha & 0 & 0 \\ 0 & 0 & 1 & -152 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Drill Module

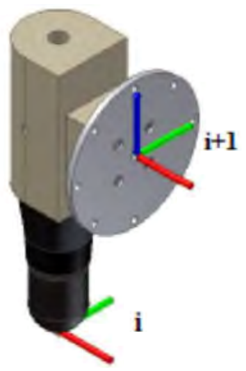


Figure A105: Drill Module i and i+1 Reference Points

Table A26: Drill Module Design Data

Offset		Euler Angles	
X	-75	γ	0
Y	0	β	0
Z	-81.5	α	0
Range of Motion	N/A	Initial Position	N/A

$$ {}^{i+1}_{i}M_{Drill} = \begin{bmatrix} 1 & 0 & 0 & -75 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & -81.5 \\ 0 & 0 & 0 & 1 \end{bmatrix} $$

Modular Range Extension Arm

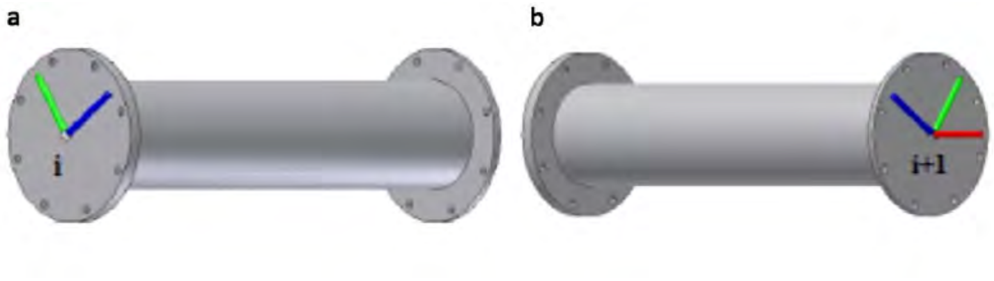


Figure A106: Modular Range Extension Arm (a) i and (b) i+1 Reference Points

Table A27: Modular Range Extension Arm Design Data

Offset		Euler Angles	
X	-370	γ	0
Y	0	β	0
Z	0	α	0
Range of Motion	N/A	Initial Position	N/A

$$ {}^{i+1}_{i}M_{Ext\ Arm} = \begin{bmatrix} 1 & 0 & 0 & -370 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} $$

Appendix B: *Distributed Module Schematics*

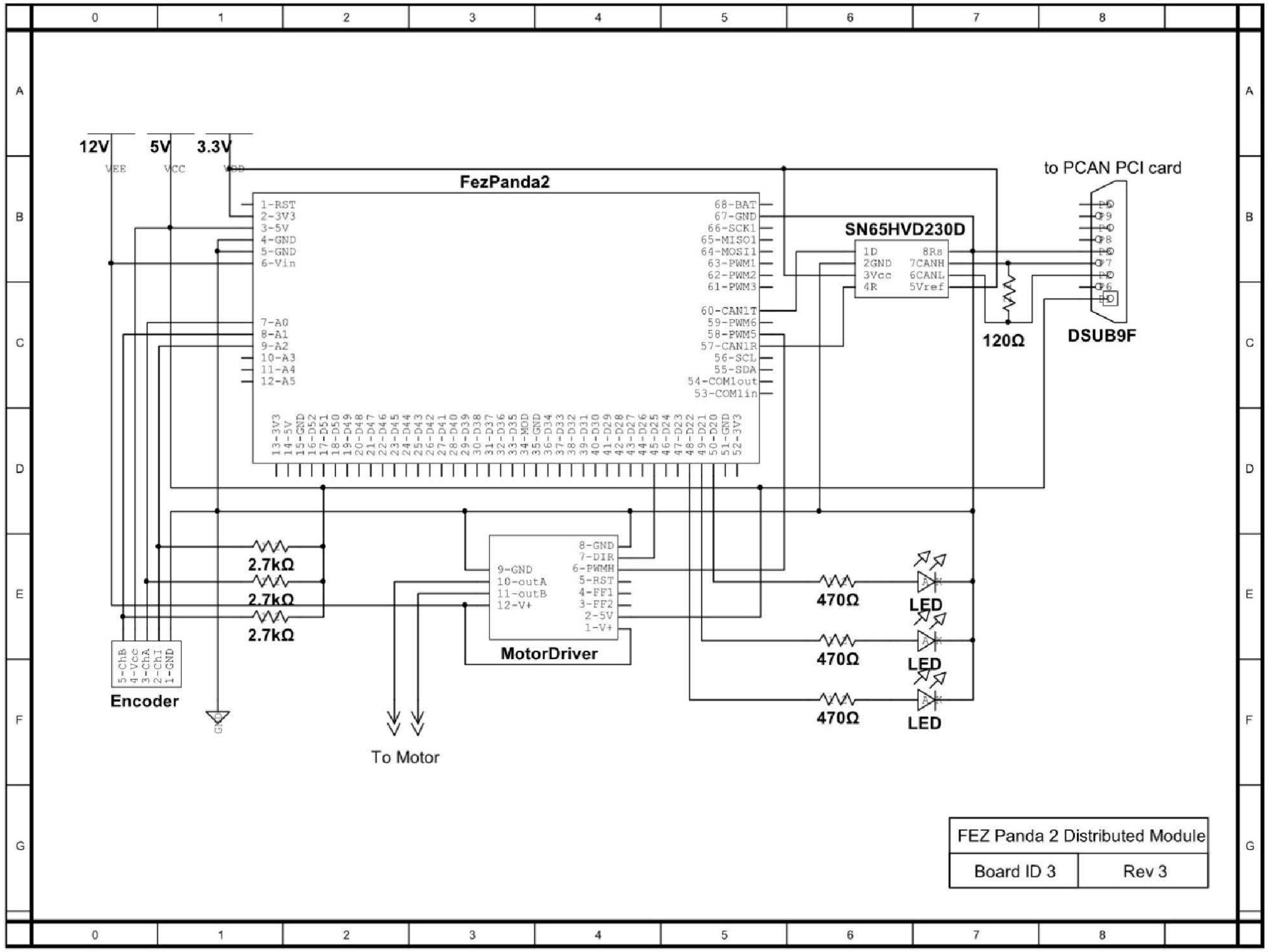


Figure B107: Board ID 3 Schematic

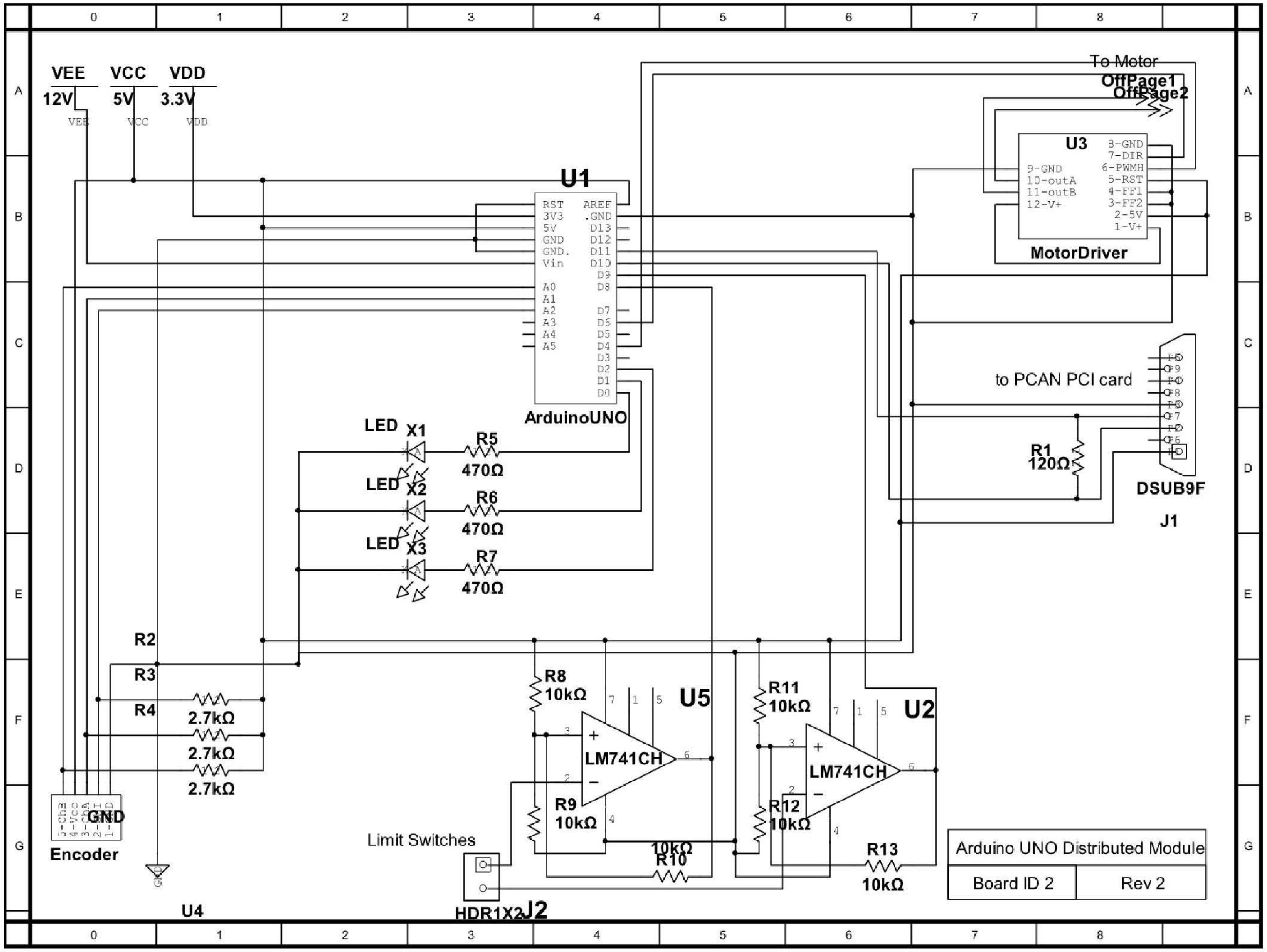


Figure B108: Board ID 2 Schematic

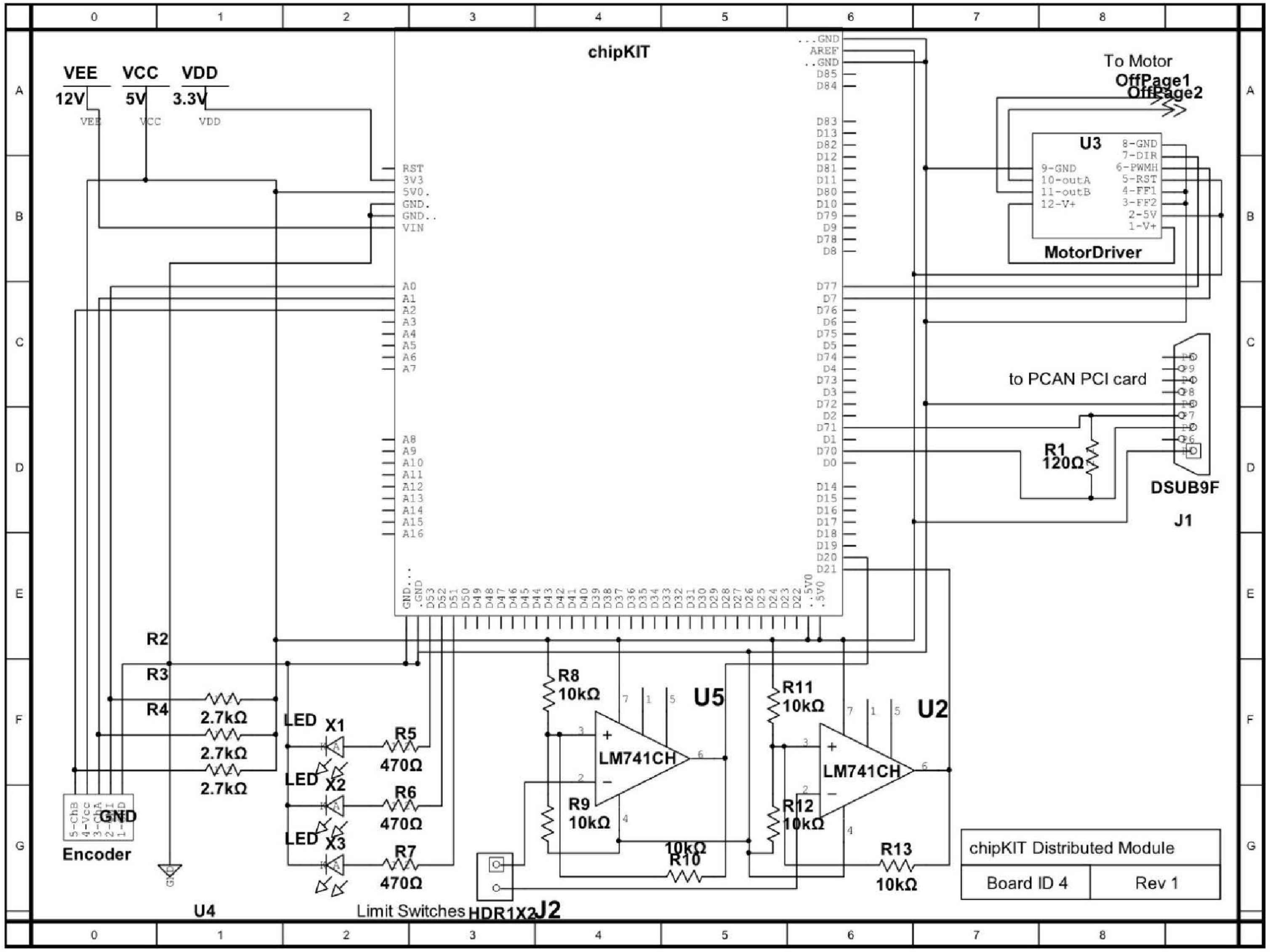


Figure B109: Board ID 4 Schematic

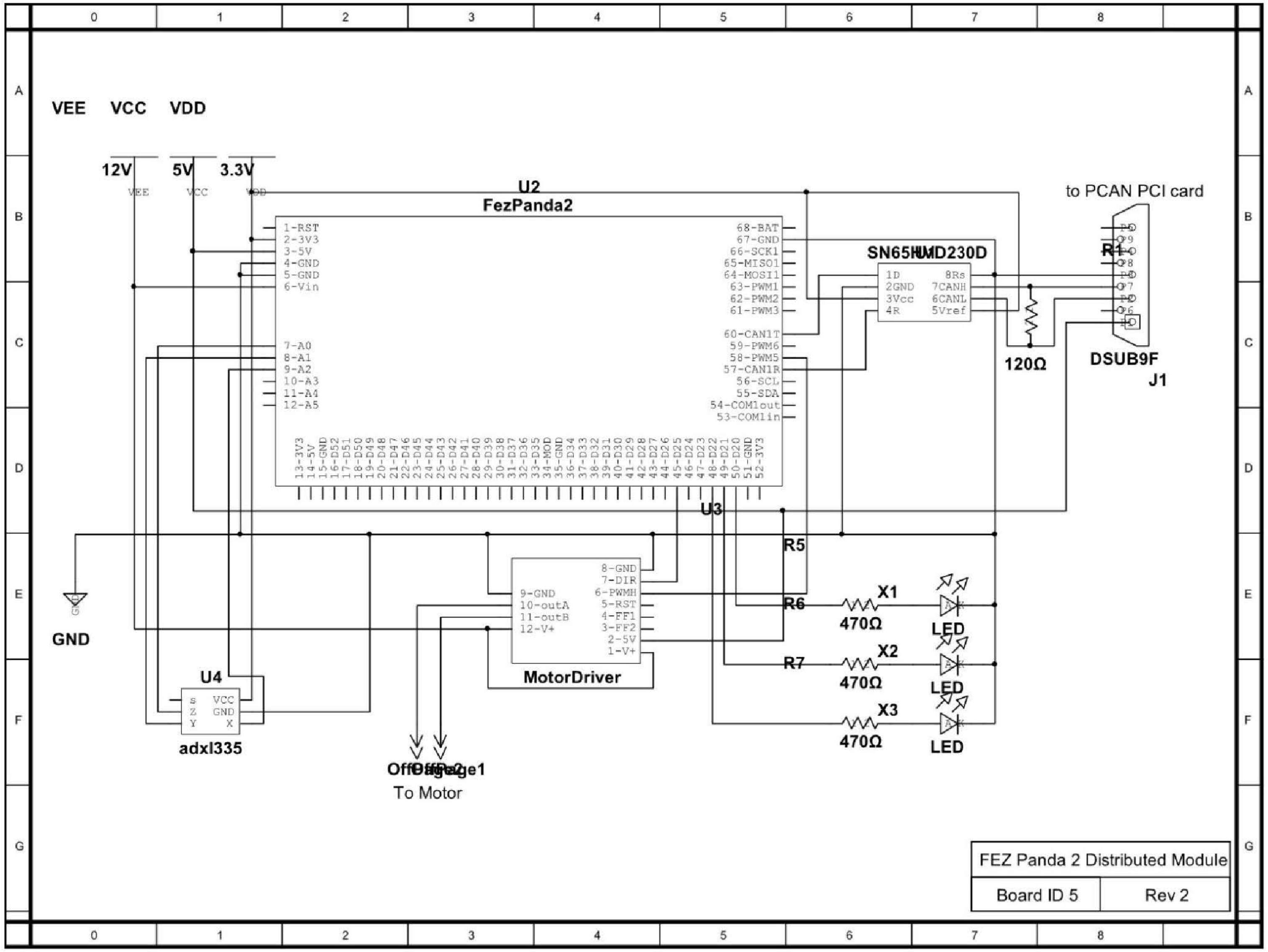


Figure B110: Board ID 5 Schematic

Appendix C: PID Code

The following sample code shows the discrete time implementation of the PID control routine called during axis movement, as well as the function called to update the three gains.

```
Static double Kp = 15/10;
Static double Ki = 15/10;
Static double Kd = 10/10;
Static double IterationCount = 0;
Double PID_run(int AcutalPos, int DesiredPos)
{
    double err;
    double d_err;
    double P_Out;
    double I_Out;
    double D_Out;
    double Output;

Static double int_err;
Static double last_err;

err = actualPos – desiredPos;
d_err = last _err – err;
int_err = int_err + int_err;

if (int_err > 200)
{
        Int_err = 200;
}
Else if(int_err < -200)
{
        Int_err = -200;
}

P_Out = err*Kp;
I_Out = int_err*ki
D_Out = d_err*Kd;

Output = P_Out + I_Out +  D_Out;

if (Output > 255)
{
        Output = 255;
}
Else if(Output < 0)
{
        Output = 0;
}
last _err = err;

if (IterationCount == 0)
{
        Output = 255;
IterationCount++;
}
Return Output;
}
```

Gain Tuning routine

```
Void GainTuning (double KpSet, double KiSet, double KdSet)
 {
     Kp = KpSet;
     Ki = KiSet;
     Kd = KdSet;
 }
```

Appendix D: *Sample Code for Text Interpretation*

```
//Text Interpretation routine.
// function will be called and passed the text file containing the user program

#include <cstdlib>
#include <iostream>

using System.Text;
using System.IO;
using namespace std

private bool TextInterpretation(string fileName)
{
    try
    {
        string line;
        StreamReader theReader = new StreamReader(fileName, Encoding.Default);

        // read line 1
        using (theReader)
        {
              // do this loop until no more lines exist
            do
            {
                line = theReader.ReadLine();
                if (line != null)
                {
                 // split entries from the line read by checking for spaces
                  // deliniators, then send that array to DoStuff()
                    string[] entries = line.Split(' ');

                    // string[0] is the Block number
                    // string[1] is preperatory function
                    // string[2] is X axis position
                    // string[3] is Z axis position
                    // string[4] A axis movement
                    // string[5] is feed rate
                    // string[6] is spindle speed

                    if (entries.Length > 0)
                    //first check if start or stop command in preparatory function

                    switch(string[1])
                    {
                                    //start commnad
                                    case "M00":
                                    // edit global machine operation status to true
                                    MachineOperation == True;
                                    //stop command
                                    break;

                                    case "M02":
                                    MachineOperation == False;
                                    // edit global machine operation status to false
                                    break;

                                    // standard movement - linear interpolation
                                    case "G01":
                                        // call linear interpolaiton func as per below
                                    //LinearInterpolation( AcutalXPos, AcutalZPos, DesiredXPos, DesiredZPos, FeedX, FeedZ);
                                    //convert from string to double
```

```
                    LinearInterpolation( MRMTXPos, MRMTZPos, Convert.ToDouble(string[2]),
                                         Convert.ToDouble(string[3]), ServoM2.Get_Mod_Feed(),
                                         ServoM1.Get_Mod_Feed(), Convert.ToDouble(string[5]));
                    break;

                    // cw circular interpolation
                    case "G02":
                    // call circ interpolaiton func as per below
                    //if command == go2
                    //CircularInterpolation( ActualXPos, ActualZPos, DesiredXPos, DesiredZPos, FeedX, FeedZ)
                    //convert from string to double
                    CircularInterpolation( MRMTXpos, MRMTZpos, Convert.ToDouble(string[2]),
                                           Convert.ToDouble(string[3]), ServoM2.Get_Mod_Feed(),
                                           ServoM1.Get_Mod_Feed(), Convert.ToDouble(string[5]));

                    //ccw circular interpolation
                    case "G03":
                    // call circ interpolaiton func as per below
                    //if command == go2
                    //CircularInterpolation( ActualXPos, ActualZPos, DesiredXPos, DesiredZPos, FeedX, FeedZ)
                    //convert from string to double
                    double Nfeed = -1*(Convert.ToDouble(string[5]));
                    double NXFeed = -1*ServoM2Axis.Get_Mod_Feed();
                    double NZFeed = -1*ServoM1Axis.Get_Mod_Feed();
```

```
                                    CircularInterpolation( MRMTXpos, MRMTZpos, Convert.ToDouble(string[2]),
                                                           Convert.ToDouble(string[3]), NXFeed,
                                                           NZFeed, Nfeed);
                                    break;

                                    default:
                                    break;

                      // end of switch
                      }
                      /////////////////////////////////////////
                      // Similarly the rest of the progam can be read and interpreted from the text file.
                      /////////////////////////////////////////
                  }
              }
              while (line != null);

              // Finished reading
              theReader.Close();
              return true;
              }
          }

          catch (Exception e)
          {
              return false;
          }
      }
```

Appendix E: *Linear Interpolation Code*

```
//Linear Interpolation Routine

// Functions
// LinearInterpolation
// SetTipo

#include <cstdlib>
#include <iostream>

using namespace std
  //global definitions

  Static Double Tipo = 0.1;

  //Linear interpolation routine
  void LinearInterpolation(    double AcutalXPos,
  double AcutalZPos,
  double DesiredXPos,
  double DesiredZPos,
  double FeedX,
  double FeedZ
  double CommandFeed)
  {
  //variable declerations
      double deltaX = DesiredXPos - ActualXPos;
      double deltaZ = DesiredZPos - ActualZPos;

      double L = sqrt(deltaX*deltaX + deltaZ*deltaZ);

      double deltaL;
      double LowestFeed;

      double incX;
      double incZ;

```

```
        int N;

        double Xresidual;
        double Zresidual;
    //calculate lower velocity
    //also includes calcuation if onle single axis is moved
        If (FeedX < FeedZ)
        {
            LowestFeed = FeedX;
    }
        elseIf (FeedZ < FeedX)
        {
            LowestFeed = FeedZ;
    }
     elseif (FeedX = 0)
     {
            LowestFeed = FeedZ;
            }
     elseif (FeedZ = 0)
     {
            LowestFeed = FeedX;
            }
     elseif(CommandFeed < FeedZ)
     {
            LowestFeed = CommandFeed;
            }
     elseif(CommandFeed < FeedX)
     {
            LowestFeed = CommandFeed;
            }

    //calculate delta length
        deltaL = LowestFeed*Tipo;

    //calculate axial increments
        incX = (deltaL)( (deltaX) / (L) );
        incZ = (deltaL)( (deltaZ) / (L) );

    //calculate number of iterations
        N = (L/deltaL);

     //define the output array with lenght N + 1 for the residual
           double OutputArrayX[N + 1];
           double OutputArrayZ[N + 1];

    //calculate residual lenghts
        Xresidual = deltaX - N*incX;
        Zresidual = deltaZ - N*incZ;

        for (i = 0 ; i = N - 1 ; I++)
        {
            OutputArrayX[i] = incX;
            OutputArrayZ[i] = incZ;
            }

        OutputArrayX[N] = Xresidual;
        OutputArrayZ[N] = Zresidual;

        //copy data to global Arrays
        for (i = 1; i = N ; i==)
        {
        *XLinInterpArray[i] = OutputArrayX[i];
        *ZLinInterpArray[i] = OutputArrayZ[i];
        }

    //calculate feedrates for residual length and updates the variables via its pointer
        *residualFeedX = ( (60*Xresidual) / (Tipo) );
        *residualFeedZ = ( (60*Zresidual) / (Tipo) );

    //update variables via their pointers
  //  *Xresidual = Xresidual;
  //  *Yresidual = Yresidual;
```

```
    }

Void SetTipo(double T)
{
        Tipo = T;
        Return 0;
        }
```

Appendix F: Circular Interpolation Code

```
// Functions
//CircularInterpolation
//SetAlpha

#include <cstdlib>
#include <iostream>

using namespace std;

//global definitions
double CircularInterpolationAlpha;

void CircularInterpolation(
double ActualXPos,
double ActualZPos,
double DesiredXPos,
double DesiredZPos,
double FeedX,
Double FeedZ)
{
	// variable declerations
	double Alpha = CircularInterpolationAlpha;
	double Pi = 3.14159265359;

	double deltaX = DesiredXPos - ActualXPos;
	double deltaZ = DesiredZPos - ActualZPos;

	double LowestFeed;
	double Radius;
	double DS;

	double FinalVelocityX;
	double FinalVelovityZ;

	bool CCW;
```

```

        //check if feeds are negatove, then rotation is ccw
        if (FeedX < 1 && if FeedZ < 1 && CommmandFeed < 1)
        {
                FeedX = -1*FeedX;
                FeedX = -1*FeedZ;
                CommandFeed = -1*CommandFeed;
                CCW = true;
                }

        if (FeedX < FeedZ)
        {
               LowestFeed = FeedX;
        }
        elseif (FeedZ < FeedX)
        {
               LowestFeed = FeedZ;
        }
        elseif(CommandFeed < FeedZ)
        {
           LowestFeed = CommandFeed;
           }
        elseif(CommandFeed < FeedX)
        {
           LowestFeed = CommandFeed;
           }


        DS = sqrt(deltaX*deltaX + deltaZ*deltaZ);

        FinalVelocityX = ( ((LowestFeed)(deltaX)) /  DS );
        FinalVelocityZ = ( ((LowestFeed)(deltaZ)) /  DS );

        if(CCW == True)
        {
               FinalVelocityX = -1*FinalVelocityX;
```

```
                FinalVelocityY = -1*FinalVelocityY;
                }

        Radius = ( 16 / (Alpha*Alpha) );

        Double Iterations;

        Iterations = ( (Pi)/(8) )*(sqrt(Radius));

        //update variables via their pointers
        *XFeed = FinalVelocityX;
        *ZFeed = FinalVelocityZ;
        *Iterations = Iterations;
        }

Void SetAlpha(int Alpha)
{
     CircularInterpolationAlpha = Alpha;
     Return 0;
     }
```

Appendix G: *Acceleration Deceleration Code*

```
//Acceleration Deceleration Routine

// Functions
//Void AccDeccRoutine (int iterations, int Axis)
//void SetMultipliers (int N, int contant)

#include <cstdlib>
#include <iostream>

using namespace std

//global definitions
    int Multiplpiers[];

Void AccDeccRoutine (int iterations, int Axis)
{
     //get global variables

     // if axis == 1, then input is z axis
     // axis ==2, input x axis
     if (Axis == 1)
     {
                    for (i = 1; i = N ; i==)
                    {
                    double Input[N] = *ZLinInterpArray[i];
                    }
     }
     elseif (Axis == 2)
     {
                  for (i = 1; i = N ; i==)
                  {
                  double Input[N] = *XLinInterpArray[i];
                  }
     }

```

```
        // note last entry into array is residual length

        int N = sizeof(Input);
        N = N - 1;

        SetMultipliers(N, 2);

        double SumMultipliers;

        for (i = 0 ; i = N ; I++)
        {
            SumMultipliers = SumMultipliers + Multipliers[i];
            }

        double Ouput[N];

        for (i = 0 ; i = N ; I++)
        {
            Output[i] = ( ((Input[i])*(Multipliers[i])) / SumMultipliers );
            }
        }

        // copy data accross to global variable
        if (Axis == 1)
        {
                       for (i = 1; i = N ; i==)
                       {
                       *ZAccDecOutputArray[i] = Output[i];
                       }
        }
        elseif (Axis == 2)
        {
                     for (i = 1; i = N ; i==)
                     {
                      *XAccDecOutputArray[i] = Output[i];
                     }
```

```
    }

}


// set multiplier function
void SetMultipliers (int N, int contant)
{
    //set all to concstant
    for (i = 0 ; i = N ; I++)
    {
        Multipliers[N] = constant
        }

    //then modify first and last values of multiplier array
    Multipliers[0] = 1;
    Multipliers[1] = 2;
    Multipliers[2] = 3;

    Multipliers[N-2] = 3;
    Multipliers[N-1] = 2;
    Multipliers[N] = 1;
    }
```

Appendix H: *Sample Code of CANbus Class*

```
    #region Can Bus class

    class CANBusClass
    {
        public static void CB_initialize()
        {
            PCANBasic.Initialize(PCANBasic.PCAN_USBBUS1, TPCANBaudrate.PCAN_BAUD_500K);
        }
        public static void CB_Deinitialize()
        {
            PCANBasic.Uninitialize(PCANBasic.PCAN_USBBUS1);
        }
        public static void CB_Module_CALL()
        {
        TPCANMsg CANMsg;
            // We create a TPCANMsg message structure
            CANMsg = new TPCANMsg();
            CANMsg.DATA = new byte[8];
            CANMsg.ID = Convert.ToUInt32(1);
            CANMsg.LEN = Convert.ToByte(8);
            CANMsg.MSGTYPE = TPCANMessageType.PCAN_MESSAGE_STANDARD;
            // can MSG data - 10000000
            CANMsg.DATA[0] = Convert.ToByte(1);
            for (int i = 1; i < 8; i++)
            {
                CANMsg.DATA[i] = Convert.ToByte(0);
            }
            PCANBasic.Write(PCANBasic.PCAN_USBBUS1, ref CANMsg);
        }
        public static void CB_Module_Data_CALL(int[] BoardIDs, int modBoardCount)
        {
            TPCANMsg CANMsg;
            // We create a TPCANMsg message structure
            CANMsg = new TPCANMsg();
            CANMsg.DATA = new byte[8];
            CANMsg.LEN = Convert.ToByte(8);
            CANMsg.MSGTYPE = TPCANMessageType.PCAN_MESSAGE_STANDARD;
            // can MSG data - 20000000
            CANMsg.DATA[0] = Convert.ToByte(2);
            for (int i = 1; i < 8; i++)
            {
                CANMsg.DATA[i] = Convert.ToByte(0);
            }
            switch(modBoardCount)
            {
                case 1:
                    CANMsg.ID = Convert.ToUInt32(BoardIDs[0]);
                    PCANBasic.Write(PCANBasic.PCAN_USBBUS1, ref CANMsg);
                break;

                case 2:
                    CANMsg.ID = Convert.ToUInt32(BoardIDs[0]);
                    PCANBasic.Write(PCANBasic.PCAN_USBBUS1, ref CANMsg);
                    CANMsg.ID = Convert.ToUInt32(BoardIDs[1]);
                    PCANBasic.Write(PCANBasic.PCAN_USBBUS1, ref CANMsg);
                break;

                case 3:
                    CANMsg.ID = Convert.ToUInt32(BoardIDs[0]);
                    PCANBasic.Write(PCANBasic.PCAN_USBBUS1, ref CANMsg);
                    CANMsg.ID = Convert.ToUInt32(BoardIDs[1]);
```

```
                        PCANBasic.Write(PCANBasic.PCAN_USBBUS1, ref CANMsg);
                        CANMsg.ID = Convert.ToUInt32(BoardIDs[2]);
                        PCANBasic.Write(PCANBasic.PCAN_USBBUS1, ref CANMsg);
                    break;

                    case 4:
                        CANMsg.ID = Convert.ToUInt32(BoardIDs[0]);
                        PCANBasic.Write(PCANBasic.PCAN_USBBUS1, ref CANMsg);
                        CANMsg.ID = Convert.ToUInt32(BoardIDs[1]);
                        PCANBasic.Write(PCANBasic.PCAN_USBBUS1, ref CANMsg);
                        CANMsg.ID = Convert.ToUInt32(BoardIDs[2]);
                        PCANBasic.Write(PCANBasic.PCAN_USBBUS1, ref CANMsg);
                        CANMsg.ID = Convert.ToUInt32(BoardIDs[3]);
                        PCANBasic.Write(PCANBasic.PCAN_USBBUS1, ref CANMsg);
                    break;
                }
            }
            public static void CB_Module_MoveCommand(int boardID, int Command, int value, int x)
            {
                TPCANMsg CANMsg;
                // We create a TPCANMsg message structure
                CANMsg = new TPCANMsg();
                CANMsg.DATA = new byte[8];
                CANMsg.ID = Convert.ToUInt32(boardID);
                CANMsg.LEN = Convert.ToByte(8);
                CANMsg.MSGTYPE = TPCANMessageType.PCAN_MESSAGE_STANDARD;
                // can MSG data - 3 - Move Command - 0 0 0 0 0 0
                CANMsg.DATA[0] = Convert.ToByte(3);
                //stop command
                if (Command == 0)
                {
                    CANMsg.DATA[1] = Convert.ToByte(0);
                    CANMsg.DATA[2] = Convert.ToByte(0);
                    CANMsg.DATA[3] = Convert.ToByte(0);
                }
                //start command
                else if (Command == 1)
                {
                    CANMsg.DATA[1] = Convert.ToByte(1);
                    //x is 0, then value is positive
                    //x is = 1 value is negative = CW rotation
                    if (x == 1)
                    {
                        CANMsg.DATA[3] = Convert.ToByte(1);
                        if (value <= 0)
                        {
                            value = value * -1;
                        }
                    }
                    else
                    {
                        CANMsg.DATA[3] = Convert.ToByte(0);
                    }
                    CANMsg.DATA[2] = Convert.ToByte(value);
                }
                // switch direction command
                else if (Command == 2)
                {
                    CANMsg.DATA[1] = Convert.ToByte(2);
                    CANMsg.DATA[2] = Convert.ToByte(0);
                    CANMsg.DATA[3] = Convert.ToByte(0);
                };
                // rest are 0's
                for (int i = 4; i < 8; i++)
```

```
                {
                    CANMsg.DATA[i] = Convert.ToByte(0);
                }
                PCANBasic.Write(PCANBasic.PCAN_USBBUS1, ref CANMsg);
            }
        }

        #endregion
```

Appendix I: *Sample Code for Generic Servo Module Class*

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace ServoModuleNS
{
    class ServoModuleClass
    {
        private
        int Module_CB_ID;
        char Module_CB_IDchar;
        int ServoBoardCount = 0;
        string Mod_Info = "";
        string Mod_Type = "";
        double Feed = 0;
        double Speed = 0;
        double[,] TransMat = new double[4, 4];

        public ServoModuleClass(int ID)
       {
           Module_CB_ID = ID;
       }
        public void set_Module_CB_IDchar(char ID)
        {
            Module_CB_IDchar = ID;
        }
       public char Get_Module_CB_IDchar()
        {
        return Module_CB_IDchar;
```

```
             }
            public void set_Module_CB_ID(int ID)
            {
                Module_CB_ID = ID;
            }
            public int Get_Module_CB_ID()
            {
                return Module_CB_ID;
            }
             public int Get_ServoBoardCount()
             {
                 return ServoBoardCount;
             }
             public void Set_Mod_Info(string A)
             {
                 Mod_Info = A;
             }
             public string Get_Mod_Info()
             {
                 return Mod_Info;
             }
             public void Set_Mod_Type(string A)
             {
                 Mod_Type = A;
             }
             public string Get_Mod_Type()
             {
                 return Mod_Type;
             }
             public void Set_Trans_Mat(int Row, int Col, double value)
             {
                 TransMat[Row, Col] = value;
            }
            public double Get_Trans_Mat(int Row, int Col)
            {
                return TransMat[Row, Col];
            }
            public void Set_Feed(double A)
            {
                Feed = A;
            }
            public double Get_Mod_Feed()
            {
                return Feed;
            }
            public void Set_Mod_Speed(double A)
            {
                Speed = A;
            }
            public double Get_Mod_Speed()
            {
                return Speed;
            }
        }
    }
```

Appendix J: Sample Code for Physical Configuration Tab

```
        #region Physical Configuration (under hardware Modules TAB)

        private void RefreshModulesButton_Click(object sender, RoutedEventArgs e)
        {
            PConfigDiagram.Visibility = Visibility.Visible;
            label3.Visibility = Visibility.Visible;

            switch (ModBoardCount)
            {
                case 0:
                    break;
                case 1:
                    UpdateComboBox0();
                    break;
                case 2:
                    UpdateComboBox0();
                    UpdateComboBox1();
                    break;

                case 3:
                    UpdateComboBox0();
                    UpdateComboBox1();
                    UpdateComboBox2();
                    break;

                case 4:
                    UpdateComboBox0();
                    UpdateComboBox1();
                    UpdateComboBox2();
                    UpdateComboBox3();
                    break;

                default:
                    break;
            }
            SavePConfigButton.Visibility = Visibility.Visible;
        }

        private void UpdateComboBox0()
        {
            label4.Visibility = Visibility.Visible;
            WorkToolComboBox.Visibility = Visibility.Visible;

            WorkToolComboBox.Items.Add(TMC0.Get_Module_CB_IDchar() + " " + TMC0.Get_Mod_Info());
        }

        private void UpdateComboBox1()
        {
            label5.Visibility = Visibility.Visible;
            Module1ComboBox.Visibility = Visibility.Visible;

            WorkToolComboBox.Items.Add(TMC1.Get_Module_CB_IDchar() + " " + TMC1.Get_Mod_Info());

            Module1ComboBox.Items.Add(TMC0.Get_Module_CB_IDchar() + " " + TMC0.Get_Mod_Info());
            Module1ComboBox.Items.Add(TMC1.Get_Module_CB_IDchar() + " " + TMC1.Get_Mod_Info());
        }
        private void UpdateComboBox2()
        {
            label6.Visibility = Visibility.Visible;
            Module2ComboBox.Visibility = Visibility.Visible;

            WorkToolComboBox.Items.Add(TMC2.Get_Module_CB_IDchar() + " " + TMC2.Get_Mod_Info());

            Module1ComboBox.Items.Add(TMC2.Get_Module_CB_IDchar() + " " + TMC2.Get_Mod_Info());

            Module2ComboBox.Items.Add(TMC0.Get_Module_CB_IDchar() + " " + TMC0.Get_Mod_Info());
            Module2ComboBox.Items.Add(TMC1.Get_Module_CB_IDchar() + " " + TMC1.Get_Mod_Info());
            Module2ComboBox.Items.Add(TMC2.Get_Module_CB_IDchar() + " " + TMC2.Get_Mod_Info());
        }
        private void UpdateComboBox3()
        {
            label7.Visibility = Visibility.Visible;
            Module3ComboBox.Visibility = Visibility.Visible;

            WorkToolComboBox.Items.Add(TMC3.Get_Module_CB_IDchar() + " " + TMC3.Get_Mod_Info());

            Module1ComboBox.Items.Add(TMC3.Get_Module_CB_IDchar() + " " + TMC3.Get_Mod_Info());

            Module2ComboBox.Items.Add(TMC3.Get_Module_CB_IDchar() + " " + TMC3.Get_Mod_Info());

            Module3ComboBox.Items.Add(TMC0.Get_Module_CB_IDchar() + " " + TMC0.Get_Mod_Info());
            Module3ComboBox.Items.Add(TMC1.Get_Module_CB_IDchar() + " " + TMC1.Get_Mod_Info());
            Module3ComboBox.Items.Add(TMC2.Get_Module_CB_IDchar() + " " + TMC2.Get_Mod_Info());
            Module3ComboBox.Items.Add(TMC3.Get_Module_CB_IDchar() + " " + TMC3.Get_Mod_Info());
        }
        private void SavePConfigButton_Click(object sender, RoutedEventArgs e)
        {
            label12.Visibility = Visibility.Visible;

            label28.Visibility = Visibility.Visible;

            CONTROLLERlabel.Visibility = Visibility.Visible;
            PIlabel.Visibility = Visibility.Visible;
            PIDlabel.Visibility = Visibility.Visible;
```

```
                ResetControllersButton.Visibility = Visibility.Visible;

                switch (ModBoardCount)
                {
                    case 1:
                        UpdateModClasses0();
                        break;
                    case 2:
                        UpdateModClasses0();
                        UpdateModClasses1();
                        break;
                    case 3:
                        UpdateModClasses0();
                        UpdateModClasses1();
                        UpdateModClasses2();
                        break;
                    case 4:
                        UpdateModClasses0();
                        UpdateModClasses1();
                        UpdateModClasses2();
                        UpdateModClasses3();
                        break;

                }
            }

            private void UpdateModClasses0()
            {
                label8.Visibility = Visibility.Visible;
                ToolHeadModule.Visibility = Visibility.Visible;
                THRadioButton.Visibility = Visibility.Visible;
```

```

                   label24.Visibility = Visibility.Visible;
                   THradioButtonPI.Visibility = Visibility.Visible;
                   THradioButtonPID.Visibility = Visibility.Visible;

                   switch (WorkToolComboBox.SelectedIndex)
                   {
                       case 0:
                           ToolHM1.Set_Mod_Info(TMC0.Get_Mod_Info());
                           ToolHM1.set_Module_CB_IDchar(TMC0.Get_Module_CB_IDchar());
                           ToolHM1.set_Module_CB_ID(TMC0.Get_Module_CB_ID());
                           break;
                       case 1:
                           ToolHM1.Set_Mod_Info(TMC1.Get_Mod_Info());
                           ToolHM1.set_Module_CB_IDchar(TMC1.Get_Module_CB_IDchar());
                           ToolHM1.set_Module_CB_ID(TMC1.Get_Module_CB_ID());
                           break;
                       case 2:
                           ToolHM1.Set_Mod_Info(TMC2.Get_Mod_Info());
                           ToolHM1.set_Module_CB_IDchar(TMC2.Get_Module_CB_IDchar());
                           ToolHM1.set_Module_CB_ID(TMC2.Get_Module_CB_ID());
                           break;
                       case 3:
                           ToolHM1.Set_Mod_Info(TMC3.Get_Mod_Info());
                           ToolHM1.set_Module_CB_IDchar(TMC3.Get_Module_CB_IDchar());
                           ToolHM1.set_Module_CB_ID(TMC3.Get_Module_CB_ID());
                           break;
                       default:
                           break;
                   }
                   ToolHeadModule.Content = (ToolHM1.Get_Module_CB_IDchar() + " " +
ToolHM1.Get_Mod_Info());
```

```
                BoardIDs[0] = update_boardIDs(ToolHM1.Get_Module_CB_IDchar());
                BoardIDsASCII[0] = ToolHM1.Get_Module_CB_ID();
            }
        // similarly, the other update mod class functions will follow similar methods
            #endregion
```

Appendix K: *Sample Code for Data Download Tab*

```
            #region Data Download TAB

            private void RefreshAvailableModulesButton_Click(object sender, RoutedEventArgs e)
            {
                switch (ModBoardCount)
                {
                    case 1:
                        _DownlaodedItems.Add(ToolHM1.Get_Module_CB_IDchar() + " " + ToolHM1.Get_Mod_Type());
                        _MotorControlItems.Add(ToolHM1.Get_Module_CB_IDchar());

                        break;

                    case 2:
                        _DownlaodedItems.Add(ToolHM1.Get_Module_CB_IDchar() + " " + ToolHM1.Get_Mod_Type());
                        _DownlaodedItems.Add(ServoM1.Get_Module_CB_IDchar() + " " + ServoM1.Get_Mod_Type());
                        _MotorControlItems.Add(ToolHM1.Get_Module_CB_IDchar());
                        _MotorControlItems.Add(ServoM1.Get_Module_CB_IDchar());
                        break;

                    case 3:
                        _DownlaodedItems.Add(ToolHM1.Get_Module_CB_IDchar() + " " + ToolHM1.Get_Mod_Type());
                        _DownlaodedItems.Add(ServoM1.Get_Module_CB_IDchar() + " " + ServoM1.Get_Mod_Type());
                        _DownlaodedItems.Add(ServoM2.Get_Module_CB_IDchar() + " " + ServoM2.Get_Mod_Type());
                        _MotorControlItems.Add(ToolHM1.Get_Module_CB_IDchar());
```

```
          _MotorControlItems.Add(ServoM1.Get_Module_CB_IDchar());
          _MotorControlItems.Add(ServoM2.Get_Module_CB_IDchar());
          break;

        case 4:
          _DownlaodedItems.Add(ToolHM1.Get_Module_CB_IDchar() + " " + ToolHM1.Get_Mod_Type());
          _DownlaodedItems.Add(ServoM1.Get_Module_CB_IDchar() + " " + ServoM1.Get_Mod_Type());
          _DownlaodedItems.Add(ServoM2.Get_Module_CB_IDchar() + " " + ServoM2.Get_Mod_Type());
          _DownlaodedItems.Add(ServoM3.Get_Module_CB_IDchar() + " " + ServoM3.Get_Mod_Type());
          _MotorControlItems.Add(ToolHM1.Get_Module_CB_IDchar());
          _MotorControlItems.Add(ServoM1.Get_Module_CB_IDchar());
          _MotorControlItems.Add(ServoM2.Get_Module_CB_IDchar());
          _MotorControlItems.Add(ServoM3.Get_Module_CB_IDchar());
          break;

      DownloadedDataListBox.ItemsSource = _DownlaodedItems;
      MotorControlListBox.ItemsSource = _MotorControlItems;
    }

    private void AvailableModuleListBox_SelectionChanged(object sender, SelectionChangedEventArgs e)
    {
      DownloadData4ModuleButton.Visibility = Visibility.Visible;
    }

    private void DownloadData4AllModules_Click(object sender, RoutedEventArgs e)
    {
      label2.Visibility = Visibility.Visible;
```

```
                DownloadedDataListBox.Visibility = Visibility.Visible;

                CANBusClass.CB_Module_Data_CALL(BoardIDs, ModBoardCount);

                ResetAvailableModulesButton.IsEnabled = true;
            }

            private void DownloadData4ModuleButton_Click(object sender, RoutedEventArgs e)
            {
                label2.Visibility = Visibility.Visible;
                DownloadedDataListBox.Visibility = Visibility.Visible;
            }

            #endregion
```

Appendix L: Sample Code for Motor Control Tab

```
        #region Motor Control Tab

        private void MCStartButton_Click_1(object sender, RoutedEventArgs e)
        {
            //test code to determine if we kno what is selected
            //MCTestLabel.Content = 
Convert.ToInt32(MotorControlListBox.SelectedItem.ToString());

            if (radioButtonCCW.IsChecked == true)
            {
                // 0 at the end is for CCW
CANBusClass.CB_Module_MoveCommand(Convert.ToInt32(MotorControlListBox.SelectedItem.ToString()), 1, 
Convert.ToInt32(MCDistance.Text), 0);
            }
            else if (radioButtonCW.IsChecked == true)
            {
                // 1 at the is for CW
CANBusClass.CB_Module_MoveCommand(Convert.ToInt32(MotorControlListBox.SelectedItem.ToString()), 1, 
Convert.ToInt32(MCDistance.Text), 1);
            }

            //CANBusClass.
        }

        private void MCStopButton_Click_1(object sender, RoutedEventArgs e)
        {
            int x = 0;
CANBusClass.CB_Module_MoveCommand(Convert.ToInt32(MotorControlListBox.SelectedItem.ToString()), 0, 
Convert.ToByte(MCDistance.Text, 16), x);
        }

        private void MCSwitchButton_Click_1(object sender, RoutedEventArgs e)
        {
            int x = 0;
CANBusClass.CB_Module_MoveCommand(Convert.ToInt32(MotorControlListBox.SelectedItem.ToString()), 2, 
Convert.ToByte(MCDistance.Text, 16), x);
        }

        #endregion
```

Appendix M: *Reconfiguration Time Data*

Table M28: Reconfiguration Times for 3 Modules (Familiar User)

Iteration	Familiar User: 3 Modules (sec)	Familiar User: 4 Modules (sec)
1	210	223
2	231	225
3	194	210
4	187	307
5	175	238

Table M29: Reconfiguration Time for 3 Modules (New User)

Iteration	New User: 3 Modules (sec)	New User: 4 Modules (sec)
1	302	354
2	307	425
3	371	537
4	462	394
5	388	489

Appendix N: *Acceleration/Deceleration Control Example Data*

Table N30: Acceleration/Deceleration Results (Constant Multiplier Values)

Interpolator Routine Results			Acceleration/Deceleration Routine Results			
Sampling Time (k):	Interpolation Output		Sampling Time (k):	Input pulse $\Delta X(\boldsymbol{k})$	Adder output	Output Pulse $\Delta X_0\ (\boldsymbol{k})$
1	5		1	5	5	1
2	5		2	5	10	2
3	5		3	5	15	3
4	5		4	5	20	4
5	5		5	5	25	5
6	5		6	0	25	5
7	5		7	0	25	5
8	5		8	0	25	5
9	0		9	0	20	4
10	0		10	0	15	3
11	0		11	0	10	2
12	0		12	0	5	1

Table N31: Acceleration/Deceleration Results (Varied Multiplier Values)

Interpolator Routine Results			Acceleration/Deceleration Routine Results			
Sampling Time (k):	Interpolation Output		Sampling Time (k):	Input pulse $\Delta X(\boldsymbol{k})$	Adder output	Output Pulse $\Delta X_0\ (\boldsymbol{k})$
1	5		1	5	5	0.56
2	5		2	5	15	1.67
3	5		3	5	30	3.33
4	5		4	5	40	4.44
5	5		5	5	45	5
6	5		6	0	45	5
7	5		7	0	45	5
8	5		8	0	45	5
9	0		9	0	40	4.44
10	0		10	0	30	3.33
11	0		11	0	15	1.67
12	0		12	0	5	0.56

Appendix O: *Encoder Count Data*

Table O32: Encoder Count - X Axis (80mm Movement)

Iteration	CW	CCW
1	13521	13552
2	13483	13519
3	13591	13534
4	13560	13566
5	13527	13521
6	13510	13524
Average	13533	13537

Table O33: Encoder Count - Z Axis (50mm Movement)

Iteration	CW	CCW
1	8605	8529
2	8454	8455
3	8259	8399
4	8516	8526
5	8359	8375
Average	8438	8456

Table O34: Encoder Count - A Axis (345° Movement)

Iteration	CW	CCW
1	11362	11424
2	11433	11359
3	11398	11428
4	11428	11375
5	11426	11381
Average	11409	11393

Time Comparisons Data

Table O35: Time (sec) Comparisons for Movement on All Axes

Rotary		Linear			
A Axis		X Axis		Z Axis	
360 degrees		80mm		50mm	
CW	CCW	CW	CCW	CW	CCW
69,1229	56,9146	36,937	30,1213	27,7659	21,907
69,0258	55,7145	36,6825	29,3837	26,9468	22,1929
68,6396	55,3225	37,1531	30,4588	26,6028	22,1069
68,3864	55,3446	36,7625	31,0331	27,2334	21,8355
68,2104	54,8826	36,4346	29,4507	26,9866	22,3045
Average	**Average**	**Average**	**Average**	**Average**	**Average**
68,68	55,64	36,92	30,09	27,11	22,07

Appendix P: *Vibration Intensity Recording Data*

Table P36: Vibration Intensity Due to Axis Movement and Drill Operation

Count	X	Y	Z	Count	X	Y	Z
1	404	522	499	**29**	405	522	501
2	404	522	499	**30**	404	523	499
3	404	522	499	**31**	404	523	500
4	404	522	499	**32**	404	524	499
5	404	522	499	**33**	405	522	498
6	410	521	492	**34**	406	522	501
7	409	517	498	**35**	405	521	498
8	395	514	499	**36**	404	522	499
9	400	523	504	**37**	407	520	503
10	400	510	496	**38**	407	522	505
11	401	522	500	**39**	404	519	497
12	406	523	497	**40**	399	515	503
13	403	519	499	**41**	403	521	486
14	397	521	499	**42**	403	519	496
15	398	511	497	**43**	395	522	512
16	404	523	493	**44**	403	520	498
17	400	517	508	**45**	399	520	498
18	410	525	510	**46**	411	519	497
19	399	522	497	**47**	406	519	496
20	404	522	499	**48**	399	525	473
21	405	524	499	**49**	407	519	498
22	406	522	499	**50**	399	516	497
23	405	522	501	**51**	397	525	495
24	404	522	499	**52**	404	523	496
25	404	523	500	**53**	396	521	493
26	404	523	498	**54**	400	521	504
27	405	524	498	**55**	406	522	497
28	406	523	499	**56**	407	526	498

Table P37: Vibration Intensity During Drilling Operation – Drill In/Out 20mm

Count	X	Y	Z	Count	X	Y	Z
1	403	523	499	**82**	417	524	518
2	403	523	499	**83**	389	526	502
3	404	524	499	**84**	427	508	514
4	403	523	499	**85**	458	520	506
5	403	523	499	**86**	382	522	488
6	404	524	499	**87**	400	513	491
7	403	523	499	**88**	408	526	535
8	403	524	498	**89**	393	519	506
9	403	523	498	**90**	413	523	495

10	403	523	499	**91**	431	526	478
11	403	524	499	**92**	396	516	482
12	403	523	499	**93**	395	511	465
13	403	523	499	**94**	392	516	514
14	401	524	501	**95**	381	514	514
15	404	523	493	**96**	431	497	538
16	405	524	480	**97**	439	522	533
17	400	521	519	**98**	396	524	516
18	401	518	507	**99**	405	512	458
19	408	530	486	**100**	396	516	490
20	409	522	502	**101**	412	528	474
21	400	515	496	**102**	403	520	552
22	406	526	497	**103**	418	541	518
23	403	526	499	**104**	399	522	535
24	408	522	493	**105**	388	518	461
25	404	526	502	**106**	418	523	508
26	406	525	504	**107**	395	510	501
27	403	528	493	**108**	426	532	436
28	401	521	513	**109**	412	528	507
29	405	524	504	**110**	406	519	523
30	399	523	503	**111**	417	520	542
31	411	501	534	**112**	402	522	504
32	403	520	498	**113**	400	526	472
33	406	523	503	**114**	398	495	497
34	406	516	507	**115**	418	517	523
35	399	515	489	**116**	397	511	524
36	400	527	496	**117**	405	521	516
37	399	525	504	**118**	389	523	514
38	398	518	508	**119**	410	521	512
39	408	529	506	**120**	406	516	480
40	402	519	494	**121**	404	511	505
41	398	520	480	**122**	417	528	494
42	396	530	496	**123**	410	529	503
43	406	521	516	**124**	405	522	490
44	401	522	496	**125**	393	507	500
45	398	518	529	**126**	409	512	520
46	395	515	511	**127**	398	513	515
47	398	533	489	**128**	422	529	496
48	409	521	510	**129**	402	532	500
49	399	525	499	**130**	415	513	499
50	390	507	507	**131**	431	532	500
51	407	519	535	**132**	415	515	489
52	412	536	486	**133**	418	523	509
53	405	520	509	**134**	402	524	505
54	407	520	537	**135**	405	526	495

55	432	535	500	**136**	402	518	503
56	395	517	510	**137**	401	506	525
57	406	523	480	**138**	407	523	495
58	403	520	476	**139**	402	518	507
59	410	518	487	**140**	399	525	503
60	404	537	477	**141**	405	523	491
61	399	513	510	**142**	402	514	497
62	408	501	537	**143**	411	518	503
63	395	520	503	**144**	401	518	488
64	390	519	492	**145**	407	512	502
65	403	503	528	**146**	408	518	494
66	405	518	479	**147**	406	523	500
67	396	519	531	**148**	406	526	493
68	414	511	494	**149**	404	519	502
69	398	499	505	**150**	405	522	507
70	408	525	526	**151**	401	525	500
71	403	518	497	**152**	405	525	500
72	412	516	525	**153**	405	514	506
73	433	552	521	**154**	403	523	499
74	419	523	481	**155**	403	523	499
75	398	533	478	**156**	404	524	499
76	401	524	511	**157**	403	523	499
77	411	518	521	**158**	403	523	499
78	396	507	463	**159**	404	524	499
79	417	528	520	**160**	403	523	499
80	393	527	499	**161**	403	524	498
81	407	514	515	**162**	403	523	498

Table P38: Vibration Intensity During Drilling Operation – Drill In/Out 40mm

Count	**X**	**Y**	**Z**	**Count**	**X**	**Y**	**Z**
1	402	520	502	**128**	387	511	530
2	400	521	505	**129**	440	513	436
3	402	521	502	**130**	418	497	510
4	402	521	503	**131**	423	544	473
5	405	520	502	**132**	431	483	485
6	402	521	502	**133**	406	513	473
7	398	520	502	**134**	422	517	520
8	402	521	502	**135**	428	526	556
9	403	520	502	**136**	404	489	519
10	402	520	502	**137**	411	483	513
11	402	521	502	**138**	397	551	515
12	402	521	502	**139**	383	512	534
13	402	521	502	**140**	411	467	508
14	402	520	502	**141**	388	558	502

15	403	520	502	**142**	397	489	502
16	396	505	530	**143**	450	473	521
17	403	522	501	**144**	438	538	498
18	410	506	507	**145**	414	533	496
19	407	523	509	**146**	407	461	496
20	415	517	499	**147**	423	517	522
21	398	518	503	**148**	402	544	474
22	396	517	502	**149**	432	474	489
23	403	518	507	**150**	442	482	513
24	404	515	508	**151**	421	553	470
25	404	515	499	**152**	406	482	507
26	402	523	502	**153**	394	464	496
27	399	498	519	**154**	397	566	481
28	406	518	502	**155**	430	491	520
29	392	517	483	**156**	440	426	506
30	406	512	513	**157**	406	540	501
31	398	524	497	**158**	397	502	483
32	397	505	499	**159**	442	513	518
33	401	515	501	**160**	481	497	482
34	396	505	487	**161**	400	505	473
35	406	517	496	**162**	404	498	495
36	403	510	517	**163**	405	472	472
37	403	517	526	**164**	391	499	484
38	406	528	506	**165**	416	507	501
39	405	525	521	**166**	417	475	475
40	388	511	506	**167**	402	518	496
41	403	537	528	**168**	390	496	500
42	401	524	497	**169**	401	490	508
43	401	518	526	**170**	436	500	531
44	409	523	509	**171**	417	502	525
45	404	498	495	**172**	398	494	503
46	395	512	530	**173**	439	468	491
47	411	520	503	**174**	408	503	519
48	405	515	537	**175**	412	455	494
49	403	511	516	**176**	420	503	523
50	413	524	499	**177**	394	494	499
51	412	521	522	**178**	436	477	520
52	378	501	507	**179**	412	488	493
53	409	528	488	**180**	401	503	518
54	430	533	502	**181**	477	495	492
55	414	515	511	**182**	399	490	529
56	408	527	494	**183**	422	479	490
57	409	524	505	**184**	434	483	495
58	397	502	500	**185**	406	491	482
59	409	513	487	**186**	406	530	524

60	401	510	482	**187**	386	514	490
61	425	518	485	**188**	437	524	483
62	413	520	474	**189**	395	525	481
63	416	496	442	**190**	401	468	496
64	407	508	490	**191**	402	525	506
65	411	521	501	**192**	405	516	477
66	392	500	500	**193**	419	516	493
67	413	511	508	**194**	412	517	487
68	411	524	443	**195**	394	511	473
69	402	546	539	**196**	401	507	481
70	436	532	507	**197**	384	529	522
71	428	490	542	**198**	408	538	510
72	408	487	486	**199**	403	505	508
73	414	502	512	**200**	393	500	514
74	441	488	540	**201**	403	537	498
75	405	506	482	**202**	396	512	518
76	433	506	520	**203**	394	528	544
77	434	486	500	**204**	410	533	473
78	416	501	431	**205**	393	522	555
79	405	494	494	**206**	405	523	517
80	387	507	485	**207**	384	494	514
81	415	469	490	**208**	410	518	522
82	438	487	514	**209**	408	508	507
83	431	481	510	**210**	400	536	482
84	403	498	461	**211**	417	526	493
85	388	488	493	**212**	395	517	487
86	402	531	527	**213**	396	535	505
87	417	482	479	**214**	407	512	503
88	416	477	490	**215**	398	520	507
89	408	509	510	**216**	394	511	497
90	402	506	525	**217**	421	505	518
91	417	534	474	**218**	410	518	516
92	398	476	463	**219**	403	520	486
93	398	501	489	**220**	398	527	495
94	438	444	503	**221**	393	508	521
95	480	464	486	**222**	404	508	505
96	417	467	493	**223**	416	520	491
97	463	471	509	**224**	405	514	520
98	407	528	542	**225**	401	527	503
99	435	492	511	**226**	413	525	481
100	401	521	472	**227**	414	524	482
101	451	475	528	**228**	411	518	507
102	404	500	491	**229**	408	508	534
103	414	486	506	**230**	392	520	489
104	437	477	484	**231**	400	499	503

105	390	505	481	**232**	421	514	513
106	443	467	516	**233**	399	523	496
107	402	510	532	**234**	403	524	504
108	391	498	558	**235**	412	521	500
109	404	524	483	**236**	396	514	493
110	419	537	422	**237**	414	535	504
111	401	474	487	**238**	409	525	490
112	398	508	453	**239**	413	524	502
113	407	486	488	**240**	414	522	505
114	431	487	459	**241**	398	508	509
115	437	481	510	**242**	399	512	504
116	423	509	502	**243**	410	518	511
117	393	461	485	**244**	409	529	508
118	439	486	505	**245**	400	517	493
119	409	524	514	**246**	400	512	502
120	402	502	502	**247**	404	527	495
121	425	515	458	**248**	402	520	502
122	386	531	511	**249**	402	521	502
123	408	527	493	**250**	402	521	502
124	394	518	477	**251**	402	521	502
125	405	518	475	**252**	402	520	502
126	402	521	530	**253**	402	521	502
127	413	530	449	**254**	402	520	502
				255	402	521	502

Appendix Q: Current Consumption Test Data

Table Q39: Current Consumption (A) - A Axis

Reading	CW	CCW
1	3,22	3,9
2	1,95	2,3
3	1,67	2,1
4	1,86	2,02
5	1,88	1,99
6	1,86	1,96
7	1,78	1,95
8	1,81	1,91
9	1,76	1,8
10	1,78	1,81
11	1,77	1,75
12	1,76	1,78
13	1,77	1,77
14	1,74	1,76
15	1,77	1,77
16	1,8	1,82
17	1,79	1,78
18	1,75	1,76
19	1,77	1,79
20	1,81	1,82
Average	0,0905	0,091

Table Q40: Current Consumption (A) - X Axis

Reading	CW	CCW
1	4,39	5,11
2	2,89	2,35
3	2,13	3,07
4	2,98	3,04
5	2,47	2,62
6	1,89	2,54
7	2,52	3,18
8	2,83	3
9	2,43	2,49
10	1,95	2,59
11	2,54	2,74
12	2,77	3,22
13	2,24	2,7
14	1,98	2,98
15	2,66	2,33
16	2,88	3,05

17	2,27	2,51
18	2,15	3,02
19	2,58	2,39
20	2,95	2,73
21	1,97	3,08
22	2,42	2,69
23	2,63	2,34
24	2,98	2,78
25	2,67	3
26	1,94	2,35
27	2,63	2,67
28	2,84	3,04
29	2,67	2,48
30	1,98	2,33
Average	2,541	2,814

Table Q41: Current Consumption (A) - Z Axis

Reading	**CW**	**CCW**
1	6,2	4,96
2	4,85	4,35
3	4,47	3,04
4	3,7	4,83
5	3,36	4,9
6	3,48	3,88
7	3,66	3,07
8	4,78	3
9	4,56	4,09
10	3,63	3,49
11	3,46	3,55
12	3,92	3,12
13	4,86	3,11
14	4,34	4,51
15	3,92	4,07
16	3,49	3,18
17	3,42	3,07
18	4,15	3,64
19	4,89	4,54
20	4,87	3,75
21	4,16	3,15
22	3,39	3,02
23	3,52	4,17
24	3,44	4,37
25	4,82	3,49
26	3,98	3,16

27	3,35	3,12
28	3,99	4,44
29	3,51	4,11
30	3,55	3,24
31	4,65	3,09
32	4,77	3,45
Average	4	3,685

Table Q42: Current Consumption(A) Unloaded Drill

Reading	CW	CCW
1	1,48	1,4
2	1,44	1,37
3	1,42	1,36
4	1,4	1,38
5	1,38	1,36
6	1,37	1,35
7	1,39	1,36
8	1,37	1,35
9	1,39	1,38
Average	1,4	1,37

Table Q43: Current Consumption(A) During Drilling Operation

Reading	Current	Reading	Current	Reading	Current	Reading	Current
1	1,48	**26**	2,36	**51**	3,21	**76**	2,88
2	1,44	**27**	2,52	**52**	3,34	**77**	2,85
3	1,42	**28**	2,69	**53**	3,36	**78**	2,4
4	1,4	**29**	2,88	**54**	3,41	**79**	2,34
5	1,38	**30**	2,66	**55**	3,29	**80**	2,38
6	1,37	**31**	2,75	**56**	3,11	**81**	2,23
7	1,39	**32**	3,24	**57**	3,16	**82**	2,11
8	1,37	**33**	3,21	**58**	3,07	**83**	2,05
9	1,39	**34**	3,25	**59**	2,92	**84**	1,96
10	2,21	**35**	3,29	**60**	2,88	**85**	1,84
11	1,57	**36**	3,31	**61**	2,98	**86**	1,76
12	1,5	**37**	3,4	**62**	3,02	**87**	1,74
13	1,53	**38**	3,37	**63**	3,05	**88**	1,72
14	1,53	**39**	3,43	**64**	3,01	**89**	1,67
15	1,6	**40**	3,9	**65**	3,2	**90**	1,6
16	1,65	**41**	3,48	**66**	3,17	**91**	1,59
17	1,72	**42**	3,54	**67**	3,16	**92**	1,57
18	1,81	**43**	3,42	**68**	3,14	**93**	1,55
19	1,82	**44**	3,45	**69**	3,1	**94**	1,52
20	1,88	**45**	3,48	**70**	3,09	**95**	1,5

21	2,01	**46**	3,22	**71**	3,13	**96**	1,47
22	2,04	**47**	3,3	**72**	3,08	**97**	1,44
23	2,09	**48**	3,28	**73**	3,05	**98**	1,4
24	2,21	**49**	3,33	**74**	3,02	**99**	1,41
25	2,38	**50**	3,3	**75**	2,92	**100**	1,4