THE DESIGN AND IMPLEMENTATION OF A SYSTEM FOR MACHINE MONITORING

by

Ben Noble

Faculty of Technology, Design and Environment

School of Engineering, Computing and Mathematics

A thesis submitted in partial fulfilment of the requirements of Oxford Brookes University for the degree of

MSc by Research

December 2020

Oxford Brookes University

Acknowledgments

I would like to sincerely thank Dr Steve Barker, Professor Khaled Hayatleh and Dr Mohamed Ben-Esmael for their continued and unwavering support throughout the entire research degree. Their knowledge and insight helped me many times and allowed me to progress through this process knowing that I had strong support and an exemplary supervisory team with me. I also owe many thanks to Rajasekhar Nagulapalli for their constant support and encouragement.

My family, particularly my mother Jackie, my father David and my brother Lewis, were also incredibly supportive and understanding with me during this research degree. They helped me keep focussed and reassured me when I was in doubt.

Finally, my gratitude goes to all of my managers and work associates, old and new, for their professional encouragement, understanding and flexibility.

Abstract

This thesis describes the research, design, manufacturing and testing of a novel multi-sensor machine monitoring system: this system integrated with existing programmable logic controller (PLC) systems and wirelessly transmitted condition data.

The research began by exploring machine monitoring strategies and the individual sensor topic areas. Methods for interfacing with PLCs and microcontroller-based wireless transmission systems were also explored. Potential electrical and physical interferences in the system were identified alongside methods to minimise them. The designs were then simulated, prototyped and revised before final manufactured designs were practically tested on factory machines.

The Clutch and Brake (C&B) sensor measured the wear of the pads in 0.08cm windows with a minimum detectable increment of 0.02cm. The C&B sensors total range was 26cm which meant the sensor could measure to a resolution of 0.08% of its total range with a sensitivity of 0.27V/cm. The wear conditions were correctly reported to the PLC which activated the programmed corrective actions. The Vibration Analysis (VA) sensor measured vibration in different motors and highlighted mechanical looseness in a motor which had recently undergone maintenance. The bandwidth of the VA sensor was 1kHz with an amplitude of acceleration within the ±16g range. The resolution of the acceleration measurement was 3.9mg which was 0.02% of the established 20g measuring range. The conditions of the motors were determined by programming which was precise to 3dp. The Non-Intrusive Pipe Pressure (NIPP) sensor monitored the pressure inside a pipe in 10bar brackets which equated to 18mV: the entire sensor range covered 0.18V which was only 3.6% of the available microcontroller range. This allowed a clear press pressure cycle to be graphically displayed from the NIPP sensor data. A leak was simulated in the PLC program and the pumps were promptly shutdown as intended.

All the sensors wirelessly transmitted their conditions to a central graphical device which generated an accompanying graphical user interface. Practical methods utilised to minimise the effect of interferences in the system such as decoupling capacitors, data averaging and filtering ensured that the conditions were reported accurately in the factory environment; achieving the aims of the project.

The developed multi-sensor system improved the condition monitoring strategies of the factory by reducing the downtime of equipment and enabling better utilisation of maintenance hours. The automated monitoring aspect of the system – coupled with the integration with the PLC system – made it viable for wide implementation across the factory. Overall, this system identified conditions accurately, automatically and succinctly which allowed it to succeed in a working factory environment.

List of Abbreviations and Terminology

AC	Alternating Current
Accelerometer	A sensor which measured acceleration
ADC	An Analog to Digital Converter
Arduino	A microcontroller board with numerous I/O
C&B	A Clutch and Brake mechanical drive method for moving presses
CBM	Condition Based Monitoring
CGD	The Central Graphical Device which showed the status of the monitored machines
CM	Condition Monitoring was a methodology for monitoring machine wear
CMRR	The Common Mode Rejection Ratio of an IA determined how well it amplified a signal while rejecting noise
DC	Direct Current
DFT	The Discrete Fourier Transform was a mathematical method of converting time-based data into frequency-based data
EMI	Electromagnetic Interference
FFT	The Fast Fourier Transform was an algorithm which computed the Discrete Fourier transform
FIFO	First In First Out operations related to data storage and retrieval
GLN	Ground Loop Noise was electrical noise present in ground loops
GPIO	General Purpose Input Output were electrical connections for other devices to connect to and transmit data
GUI	A Graphical User Interface which used graphics to display data
High Pass Filter	An electronic filter which only allowed signals through that were after the pre-determined frequency
I2C	Inter-Integrated Circuit was a data transmission protocol
IA	Instrumentation Amplifier
IC	Integrated Circuit
IDE	Integrated Development Environment
Industry 4.0	The next industrial revolution focussed on data harvesting, analysis and wireless data transmission
I/O	Input and Output connections on electrical equipment
IoT	Internet of Things described data collecting devices connected to the cloud
IP	Ingress Protection rated enclosures for certain environmental conditions
IR	Infrared was a type of electromagnetic wave
ISM	The 2.4GHz license-free Industrial, Scientific and Medical Radio Band.

LED	Light Emitting Diode
MCB	Miniature Circuit Breaker protected electrical systems
MEMS	Micro-Electrical Mechanical Systems were used in accelerometers
NDE	Non-Drive End of a motor
NIPP	Non-Intrusive Pipe Pressure
ODR	Output Data Rate
PLC	Programmable Logic Controller
PPM	Planned Preventative Maintenance
Raspberry Pi	A brand of micro-computers that ran on a Linux variant
RF	Radio Frequency
RM	Reactive Maintenance
RMS	Root Mean Square was the division of a peak value by root 2.
RPM	Revolutions per minute
SAW	Surface Acoustic Wave
SPI	Serial Peripheral Interface
TDC	Top Dead Centre was the zero degrees position of a press
VA	Vibration Analysis

List of Figures and Diagrams

Figure 2.1: Potential Divider 2-6
Figure 2.2: Diagram of C&B Unit 2-7
Figure 2.3: Graph showing voltage, distance and reflection (Sharp, no date, p.4) 2-8
Figure 2.4: MEMS Capacitive Accelerometer (Sinha *et al.*, 2014) 2-9
Figure 2.5: IAM-20381 Accelerometer (TDKInvensense, 2017, p.1) 2-10
Figure 2.6: Motor Vibration Frequency Spectrum 2-11
Figure 2.7: SAW Sensor Layout (University of Cambridge, no date) 2-12
Figure 2.8: Mounted Metallic Strain Gauge (*ibid*, p.2) 2-13
Figure 2.9: Quarter Bridge Circuit 2-14
Figure 2.10: Strain Gauge Circuit with Thermal Correction (National Instruments, 1998) 2-15
Figure 2.11: Arduino Power Supply Circuit Schematic (*ibid*) 2-16
Figure 3.1: Output Voltage Graph for the SHARP GP2Y0A41SK0F (Sharp, no date, p.5) 3-23
Figure 3.2: Block Diagram for Clutch and Brake Sensor 3-25
Figure 3.3: Block Diagram of ADXL345 Accelerometer (Analog Devices, 2015) 3-28
Figure 3.4: VA Sensor System Block Diagram 3-32
Figure 3.5: INA125 Interior Schematic (*ibid*) 3-39
Figure 3.6: Strain and Resistance Change Graph 3-40
Figure 3.7: Voltage Output, Strain Change vs Internal Pipe Pressure Graph 3-41
Figure 3.8: Change in Voltage Output with Pressure Graph 3-41
Figure 3.9: NIPP Sensor Block Diagram 3-43
Figure 3.10: General Voltage Regulator Setup 3-44
Figure 3.11: Potential Divider for 24V to 5V 3-44
Figure 3.12: Potential Divider for 24V to 3.3V 3-45
Figure 3.13: Switching Optocoupler System 3-45
Figure 3.14: nRF24L01 Module Diagram (Components101, 2018) 3-45
Figure 3.15: nRF24L01 Breakout Board (Cytron, no date) 3-46
Figure 3.16: nRF Wireless System Block Diagram 3-46
Figure 3.17: PLC Program NW1-4 3-50
Figure 3.18: PLC Program NW5-7 3-51
Figure 3.19: PLC Program NW8 3-51
Figure 3.20: PLC Program NW9-11 3-52
Figure 3.21: PLC Program NW12-15 3-53
Figure 3.22: PLC Program NW16 3-54
Figure 3.23: PLC Program NW17 3-54
Figure 3.24: PLC Program NW18 3-55
Figure 3.25: PLC Program NW19 3-56
Figure 3.26: PLC Program NW20 3-57
Figure 3.27: PLC Program NW21-22 3-57
Figure 3.28: PLC Program NW23 3-58
Figure 3.29: PLC Program NW24 3-58
Figure 3.30: PLC Program NW25-26 3-59
Figure 3.31: PLC Program NW36-38 3-59
Figure 3.32: PLC Program NW39 3-60
Figure 3.33: PLC Program NW40 3-60
Figure 3.34: PLC Program NW41 3-61
Figure 3.35: PLC Program NW42-43 3-62
Figure 4.1: Steady State Test Results Acceleration FFT Graph 4-68
Figure 4.2: Steady State Test Results Velocity RMS FFT Graph 4-68
Figure 4.3: 2400 RPM Test Results Acceleration v Time Graph 4-69
Figure 4.4: 2400 RPM Test Results Velocity v Time Graph 4-69
Figure 4.5: 2400 RPM Test Results Detrended Velocity v Time Graph 4-70
Figure 4.6: 2400 RPM Test Results Detrended & HP Filtered Velocity v Time Graph 4-70
Figure 4.7: 2400 RPM Test Results Detrended & HP Filtered RMS Velocity v Time Graph ... 4-70

Figure 4.8: 2400 RPM Test Results Acceleration FFT Graph 4-71
Figure 4.9: 2400 RPM Test Results RMS Velocity FFT Graph 4-72
Figure 4.10: Strain Gauge and Bridge Circuit 4-73
Figure 4.11: Strain Gauge and Bridge Simulations 4-73
Figure 4.12: Prototype Test of NIPP Sensor... 4-74
Figure 4.13: 24V to 5V Divider Simulation Results 4-75
Figure 4.14: 24V to 3.3V Divider Simulation Results 4-75
Figure 4.15: Optocoupler 5V Simulation Results 4-76
Figure 4.16: Optocoupler 3.3V Simulation Results 4-76
Figure 4.17: nRF and GUI 1st Test Python Shell 4-77
Figure 4.18: Generated GUI from 1st nRF Test '1NO2OK306' 4-78
Figure 5.1: Connections Junction Box 5-80
Figure 5.2: Complete Clutch and Brake Sensor Circuitry 5-81
Figure 5.3: Sensor Mounted to C&B unit 5-81
Figure 5.4: VA Circuit Boards 5-84
Figure 5.5: Main Motor VA 5-84
Figure 5.6: Main Motor Acceleration FFT 5-85
Figure 5.7: 6000 RPM Test Results Acceleration FFT Graph 5-85
Figure 5.8: 6000 RPM Test Results RMS Velocity Graph 5-86
Figure 5.9: NIPP Circuit Boards inside Enclosure 5-87
Figure 5.10: Strain Gauge Installation 5-87
Figure 5.11: NIPP Rate of Change Graph over 2 cycles 5-89
Figure 5.12: PLC Program NW5-7 Online 5-91
Figure 5.13: PLC Program NW8 Online 5-92
Figure 5.14: PLC Program NW16 Online 5-92
Figure 5.15: PLC Program NW18 Online 5-93
Figure 5.16: PLC Program NW18 Actioned Online 5-93
Figure 5.17: PLC Program NW19 Online 5-94
Figure 5.18: PLC Program NW19 Actioned Online 5-94
Figure 5.19: PLC Program NW20 Online 5-95
Figure 5.20: PLC Program NW22 Online 5-95
Figure 5.21: PLC Programming NW40 Online 5-96
Figure 5.22: PLC Programming NW41 Online 5-96
Figure 5.23: PLC Programming NW42 Online 5-97
Figure 5.24: PLC Programming NW43 Online 5-97
Figure 5.25: CGD External View 5-98
Figure 5.26: CGD GUI '1OK2WE3ER' 5-98
Figure 5.27: CGD GUI '1LE2WE308' 5-99

List of Tables

Table 2.1: Comparison of Parameters for Wireless Transmission 2-17
Table 2.2: Microcontroller Specification Table 2-18
Table 3.1: Common Requirements for all Circuits 3-21
Table 3.2: Expected Measurements - Manual method 3-22
Table 3.3: Ranges and Categories of Wearing Pads 3-22
Table 3.4: Metric Ranges and Categories of Wearing Pads 3-22
Table 3.5: ADXL345 0x31 Register (*ibid*) 3-29
Table 3.6: ADXL345 0x2D Power Control Register (*ibid*) 3-29
Table 3.7: Known Values Table 3-33
Table 3.8: Strain Calculations with 5bar intervals of pressure 3-34
Table 3.9: Pressure, Strain and Resistance Changes 3-35
Table 3.10: Ranges of Strain and Resistance 3-35
Table 3.11: Difference in Resistance between Pressures 3-35
Table 3.12: Resistors Resistance Tolerances 3-36
Table 3.13: Sensor Calculations Overview 3-37
Table 3.14: Summary of IA Vout for different Pressure Ranges 3-38
Table 3.15: Arduino Resolution Calculations 3-38
Table 3.16: Collated Values of Strain, Pressure and Voltage Output 3-40
Table 3.17: Wireless Network Process Summary 3-47
Table 3.18: ASCII to DEC Conversion (WatElectronics, 2019) 3-47
Table 4.1: Lifebit Check C&B Test - Expected Results 4-64
Table 4.2: PLC Input Calibration Check C&B Test - Expected Results 4-64
Table 4.3: Initial Measurement C&B Test - Expected Results 4-65
Table 4.4: General Measurement C&B Test - Expected Results 4-65
Table 4.5: Lifebit Check C&B Test Results 4-66
Table 4.6: PLC Input Calibration Check C&B Test Results 4-66
Table 4.7: Initial Measurement C&B Test Results 4-66
Table 4.8: General Measurement C&B Test Results 4-67
Table 5.1: C&B Sensor Implementation Test 1 5-82
Table 5.2: C&B Sensor Implementation Test 2 5-83
Table 5.3: NIPP 0bar Pressure after Calibration 5-88
Table 5.4: NIPP Pressure over 2 cycles 5-89

Table of Contents

Acknowledgments ii
Abstract iii
List of Abbreviations and Terminology iv
List of Figures and Diagrams vi
List of Tables viii
Table of Contents ix
1 Introduction 1-1
1.1 Introduction and Justification 1-2
1.2 Industrial Relevance 1-2
1.3 Aim and Objectives 1-3
1.4 Original Contribution 1-3
1.5 Structure of the Thesis 1-3
2 Literature Review 2-4
2.1 Introduction 2-5
2.2 Condition Monitoring 2-5
2.3 Clutch and Brake Pad Wear Sensor 2-6
2.3.1 Resistive Measurement Method 2-6
2.3.2 Distance Measuring Method 2-6
2.4 Vibration Analysis Sensor 2-9
2.4.1 Microphone Measurement Method 2-9
2.4.2 Accelerometer Method 2-9
2.5 Non-Intrusive Pipe Pressure Sensor 2-12
2.5.1 Surface Acoustic Wave Method 2-12
2.5.2 Strain Gauge Method 2-12
2.6 Power, Interfacing Circuits and Wireless Transmission 2-16
2.6.1 Power Supply 2-16
2.6.2 Data Signal Conditioning 2-17
2.6.3 Wireless Transmission Methods 2-17
2.6.4 Central Graphical Device 2-18
2.6.5 Microcontrollers 2-18
2.7 Practical Limitations and Interferences 2-19
2.8 Summary 2-19
3 Specifications and Designs 3-20
3.1 Introduction 3-21
3.2 Common Requirements 3-21
3.3 Clutch and Brake Sensor 3-22
3.4 Vibration Analysis Sensor 3-27

3.5 Non-Intrusive Pipe Pressure Sensor 3-33
3.6 Power, Interfacing Circuits and Wireless Transmission 3-44
3.6.1 Power to the Sensors 3-44
3.6.2 24V Digital PLC Signal to 5V/3.3V Digital Microcontroller Signal 3-44
3.6.3 5V/3.3V Digital Microcontroller Signal to 24V Digital PLC Signal 3-45
3.6.4 Wireless Transmission 3-45
3.6.5 Microcontrollers 3-49
3.7 Programmable Logic Controller Programming 3-50
3.8 Summary 3-62
4 Simulations, Prototypes, Results and Analysis 4-63
4.1 Introduction 4-64
4.2 Clutch and Brake Sensor 4-64
4.2.1 Simulations 4-64
4.2.2 Prototype Testing 4-64
4.2.3 PLC and Sensor Incorporation Test Results 4-66
4.3 Vibration Analysis Sensor 4-68
4.3.1 Program Tests and Simulations 4-68
4.3.2 Steady State Test Results 4-68
4.3.3 2400 RPM Motor Test Results 4-69
4.4 Non-Intrusive Pipe Pressure Sensor 4-73
4.4.1 Simulations 4-73
4.4.2 Strain Gauge and Wheatstone Bridge Simulations 4-73
4.4.3 Prototype Testing 4-74
4.5 Power, Interfacing Circuits, PLC Program and Wireless Transmission 4-75
4.5.1 Power to the Sensors 4-75
4.5.2 24V Digital PLC Signal to 5V/3.3V Digital Microcontroller Signal 4-75
4.5.3 5V/3.3V Digital Microcontroller Signal to 24V Digital PLC Signal 4-76
4.5.4 PLC Program 4-77
4.5.5 Wireless Transmission and Central Graphical Device 4-77
4.6 Summary 4-78
5 Implementation, Testing and Results 5-79
5.1 Introduction 5-80
5.2 Connections Junction Box 5-80
5.3 Clutch and Brake Sensor 5-81
5.4 Vibration Analysis Sensor 5-84
5.5 Non-Intrusive Pipe Pressure Sensor 5-87
5.6 Power and PLC Interfacing 5-91
5.6.1 Power 5-91
5.6.2 PLC Interfacing 5-91
5.7 Wireless Transmission and Central Graphical Device 5-98

5.8 Summary 5-99
6 Conclusion and Future Work 6-100
6.1 Conclusion.. 6-101
6.2 Future Work.. 6-103
7 References.. 7-104
8 Appendices 8-109
8.1 A1 Clutch and Brake Arduino Program excluding nRF.. 8-110
8.2 A2 Clutch and Brake Interfacing and Calibration Circuit Schematic.. 8-115
8.3 A3 Vibration Analysis ADXL345 SPI C Program (Nagimov, 2016).. 8-116
8.4 A4 Vibration Analysis ADXL345 FFT Program 8-121
8.5 A5 Vibration Analysis Interfacing Circuit Schematic 8-129
8.6 A6 Non-Intrusive Pipe Pressure Arduino Program excluding nRF 8-130
8.7 A7 Non-Intrusive Pipe Pressure Interfacing and Calibration Circuit 8-140
8.8 A8 Non-Intrusive Pipe Pressure Measuring Circuit Schematic.. 8-142
8.9 A9 9V Regulator Circuit Schematic.. 8-143
8.10 A10 Clutch and Brake Pad Wear Test Transmission Program.. 8-144
8.11 A11 Non-Intrusive Pipe Pressure Test Transmission Program 8-145
8.12 A12 Vibration Analysis Serial Test Transmission Program 8-146
8.13 A13 Vibration Analysis Arduino Test Transmission Program 8-147
8.14 A14 Central Graphical Device Receiver Test Transmission Program.. 8-148
8.15 A15 Central Graphical Device GUI Test Transmission Program.. 8-149
8.16 A16 Siemens PLC Integration STL Program 8-152
8.17 A17 Clutch and Brake Sensor Prototype Test Result Graphs 8-161
8.18 A18 Connections Junction Box Schematics 8-167
8.19 A19 C&B Interfacing Circuit Photograph and Terminal Designations.. 8-171
8.20 A20 Complete C&B Arduino Program.. 8-172
8.21 A21 C&B Implementation Test Results 1.. 8-178
8.22 A22 C&B Implementation Test Results 2.. 8-179
8.23 A23 VA Interfacing Circuit Photograph and Terminal Designations 8-180
8.24 A24 NIPP Measuring Circuit Photograph and Terminal Designations.. 8-181
8.25 A25 NIPP Interfacing Circuit Photograph and Terminal Designations.. 8-182
8.26 A26 Complete NIPP Arduino Program.. 8-183
8.27 A27 NIPP Pressure Readings Graph 1.. 8-194
8.28 A28 NIPP Pressure Readings Graph 2.. 8-195
8.29 A29 9V Regulator Circuit Photograph and Terminal Designations.. 8-196

1 Introduction

1.1 Introduction and Justification 1-2
1.2 Industrial Relevance 1-2
1.3 Aim and Objectives 1-3
1.4 Original Contribution 1-3
1.5 Structure of the Thesis 1-3

1.1 Introduction and Justification

This research project aimed to design and implement a system for machine monitoring. Condition Monitoring (CM) was identified as a popular machine monitoring method; where variables were monitored over a period of time to check for deviations in behaviour from normal operation (Penman and Tavner, 1989). The process was useful for finding faults before they occurred and helped generate an identifiable pattern of failure for future reference.

The new system for machine monitoring was integrated into an existing control system enabling enhanced control. Automatic adjustments of press parameters and generation of local alarms were utilised alongside wireless transmission of some data to a standalone device. The Internet of Things (IoT) and Industry 4.0 were the main drivers of innovation in the manufacturing industry (Gilchrist, 2016). This research project funded by BMW Group UK Manufacturing Ltd was an important step in this direction.

1.2 Industrial Relevance

This research project was designed for use on mechanical and hydraulic presses; three parts of the machines were chosen for CM due to their operational importance. The first was a sensor to measure the Clutch and Brake (C&B) brake pad wear, the second was a sensor for Vibration Analysis (VA) on motors while the third was a sensor to provide Non-Intrusive Pipe Pressure (NIPP) readings of hydraulic pipes.

The C&B sensor was chosen because when the brake pads wore over time the ability of the press to stop within its Top Dead Centre (TDC) window reduced which eventually resulted in the press stopping and causing downtime. To alleviate this problem two adjustable parameters were identified: the braking start position and the TDC window. Adjustment of these parameters were corrective actions. The VA sensor monitored vibration in motors. The peak of vibration at key frequencies was important in determining problems with the motors. Alarms were programmed to be triggered above certain thresholds which allowed adequate time for repair. The NIPP sensor was intended to measure the internal pressure of hydraulic pipes without modification to existing pipework. Sudden losses in pressure indicated a leak in the pipes and therefore a sensor which detected this helped prevent oil losses. No accessible wireless network existed within the press shop. Therefore, in addition to local alarm generation some data was also wirelessly transmitted.

The business justification for this research project focussed on downtime reduction and economic gain. A clear economic benefit was the reduction of time that engineers spent on manual machine monitoring. Improved time management was another gain from this system because the early warnings ensured that work was planned in before full failure occurred.

1.3 Aim and Objectives

The aim of the project was to answer the following research question: Is it possible to design and implement a system for machine monitoring, establish possible interferences in the system and identify methods to minimise them.

This aim was completed through the following objectives:

- Review existing academic literature and industry developments.
- Formulate design specifications.
- Simulate and test the designs in software and record the findings.
- Refine and finalise the design of the system, including auxiliary connections.
- Simulate and test each finalised system component against the requirements and specifications, and analyse the results.
- Build and test an integrated multi-sensor system that enables automatic condition monitoring.
- Refine the design and modify the system as necessary.
- Analyse and discuss the final results, along with the limitations of the findings and potential solutions to any issues uncovered, and opportunities for future work.

1.4 Original Contribution

The original contribution demonstrated in this MSc by Research was an automated multi-sensor system for machine monitoring purposes which highlighted the condition of 3 important areas of presses. The original contribution also extended to the setup of a wireless network for data transmission for these sensors. The integration between the sensors and existing control system allowed for high levels of control not yet utilised.

1.5 Structure of the Thesis

This thesis began with an overview of the industrial relevance of the research project and the research aim. A literature review explored the themes behind the research question and sources were contrasted and compared with the knowledge gained being used for the designs. Following the designs came a chapter on the initial simulations and prototype testing results. The final two chapters covered the final tests performed in the factory and the conclusions drawn from them, with the research question observations highlighted.

2 Literature Review

2.1 Introduction 2-5

2.2 Condition Monitoring 2-5

2.3 Clutch and Brake Pad Wear Sensor 2-6

2.4 Vibration Analysis Sensor 2-9

2.5 Non-Intrusive Pipe Pressure Sensor 2-12

2.6 Power, Interfacing Circuits and Wireless Transmission 2-16

2.7 Practical Limitations and Interferences 2-19

2.8 Summary 2-19

2.1 Introduction

This section reviewed literature related to the research. Specific chapters covered: the importance of CM in industry, the C&B pad wear sensor, the VA sensor, the NIPP sensor, wireless communication methods, interfacing circuits and practical limitations.

2.2 Condition Monitoring

Two methods of CM identified were trend monitoring and condition checking. Trend monitoring continuously measured data which was analysed to determine how the machine was running. Condition checking was a discrete method whereby periodic measurements were taken and compared to known values to find problems (Neale and Woodley, 1975). Clear economic benefits of implementing a CM system were shown – particularly one which automatically operated and alerted. (Tavner *et al.*, 2008), discussed these economic benefits in comparison to the traditional maintenance methods of Reactive Maintenance (RM) and Planned-Preventative Maintenance (PPM).

Further research showed that a good CM system was a good pathway to developing a predictive maintenance strategy: being able to predict when equipment failed would have allowed for maintenance activities to be planned in advance reducing labour and downtime costs. Over a long period of time accurate predictions would have been created (Mobley, 2002).

2.3 Clutch and Brake Pad Wear Sensor

2.3.1 Resistive Measurement Method

The resistive method was based around measuring the resistance of the brake pads to determine the state of wear. The brake pads had their own resistivity (ρ) due to their material makeup. Lowrie, (2007) related resistivity to resistance as Equation 2.1:

$$R = \rho * \frac{L}{A} \qquad \text{(Eq. 2.1)}$$

Where R was resistance, ρ was resistivity, L was length and A was the cross-sectional area. As wear increased the value of L decreased, while resistivity and area remained the same which caused a decreased resistance. Figure 2.1 showed a potential divider.

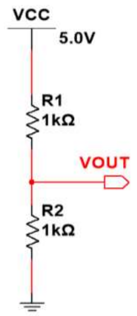


Figure 2.1: Potential Divider

If the measured resistance was substituted for R2 then its value would have changed Vout (Horowitz and Hill, 2015). A reference voltage when the brake pads were not worn could have been found and then further wear would have showed a difference in the voltage output. A comparative circuit using an op-amp comparator would have then triggered an alarm when a threshold was breached. Interferences that could have affected the results were identified. There would have been direct contact between the resistance measuring probes and the pads and so over time they would have eroded and not measured properly. Also, the pads would have increased in temperature over time due to friction which would have affected the dimensional properties and resistivity.

2.3.2 Distance Measuring Method

Observation of the C&B unit showed that the plate containing the brake pads rotated and the friction-plate did not, when the brake engaged the friction-plate moved laterally to intercept the brake pads and stop the press. The distance measurement method was based upon this observation. When the brake was engaged the gap between the two plates was measurable. The measurable distance decreased over time as showed in Figure 2.2. As time tended to infinity then the distance 'y' tended to 0 because as time passed wear increased.

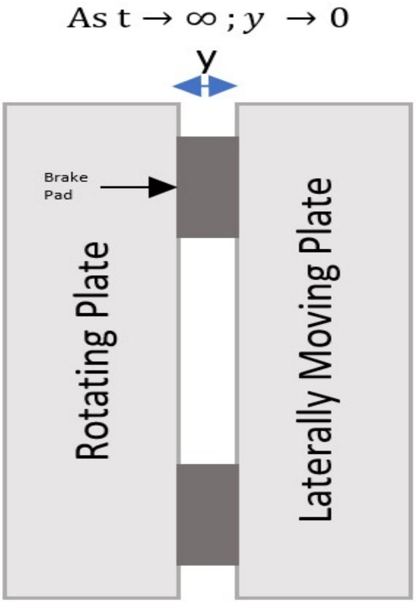


Figure 2.2: Diagram of C&B Unit

The methods researched were non-contact. High accuracy was important and the method had to be capable of operation in an environment with high levels of acoustic noise and small amounts of particulate debris. An ultrasonic sensor used ultrasonic sound waves with frequencies >20kHz (above audible range) to detect the presence of an object (Bolton, 2015). The distance was measured by counting the time taken for the ultrasonic signal to return after bouncing off the object being measured. The distance was found by multiplying the speed of ultrasonic waves and the measured time. Bolton (2015) discussed a limitation of ultrasonic sensors which was that the object the sound waves reflected off of needed to be completely flat and smooth. The particulate debris near the sensor's installation area voided this guarantee. Other considerations for ultrasonic measurement methods were discussed in a paper by (Carullo and Parvis, 2001). They mentioned how the velocity of sound in air changed with temperature, as showed in Equation 2.2.

$$V_S \approx 20.055 * \sqrt{T} \qquad \text{(Eq. 2.2)}$$

V_S was the velocity of sound while T was the temperature in Kelvin; the equation was derived from the ideal gas laws. The temperature of the air around the C&B unit would have needed to be measured which would have resulted in additional sensors.

A capacitive method was researched. Capacitive sensors as Jones (2008) described measured distance by mimicking a capacitor. The sensor consisted of two parallel plates, one was the sensor and the other was the object for distance measurement. The varying distance altered the capacitance. Equation 2.3 (Horowitz and Hill, 2015) showed the relationship:

$$C = \varepsilon * \frac{A}{d} \qquad \text{(Eq. 2.3)}$$

C was capacitance, ε was the dielectric permittivity, A was plate area and ‘d’ was the distance between the plates. The dielectric was air which had a constant value as did the area, only the distance changed. However, ‘d’ needed to be small in comparison to A (Jones, 2008) so this method was deemed unsuitable.

Infrared (IR) sensors were also researched. IR sensors were typically cheaper in cost with a faster response time than ultrasonic sensors (Mohammad, 2009). The IR sensors emitted an IR wave which reflected from the surface of the object back to the sensor. IR light was absorbed to different extents by different objects depending on their reflectance. An infrared sensor’s datasheet was inspected to see the importance of the reflectivity of the object on the output voltage from the sensor. The effect of ambient temperature was noted to have an impact on the IR sensor (*ibid*) but proper calibration ensured this impact was minimised. The Sharp GP2Y0A41SKF sensor’s datasheet (Sharp, no date), Figure 2.3, showed that objects with a reflectance ratio of 90% and 18% had little difference in output voltage.

Image removed due to 3rd party copyright.

Figure 2.3: Graph showing voltage, distance and reflection (Sharp, no date, p.4)

This property of IR sensors was considered useful because of the sensor mounting area.

2.4 Vibration Analysis Sensor

2.4.1 Microphone Measurement Method

When mechanical elements of motors began to fail vibrations were evident. The vibrations emitted acoustic waves which were monitored by piezoelectric microphones (Tandon and Choudhury, 1999) and the recorded voltages were amplified. The equipment condition was determined by the number of times the measured amplitude of vibration exceeded a threshold. Tandon and Choudhury also described a setup which used two piezoelectric microphones to detect sound wave intensity measurements in a 2-dimensional area. One disadvantage of this method was the interference caused by the high levels of background acoustic noise in the area.

2.4.2 Accelerometer Method

3 main types of accelerometer were researched: capacitive Micro-Electro Mechanical Systems (MEMS) accelerometers, piezoresistive accelerometers and piezoelectric accelerometers. MEMS capacitive accelerometers measured vibration in an object via a moving mass acting in a capacitive manner. Figure 2.4 showed a capacitive accelerometer. The proof mass moved when under vibration and the sense fingers mounted to the mass moved in relation to the fixed electrode. Different vibration levels caused changes in capacitance and hence output voltage. Capacitive accelerometers were particularly suited to low-frequency signals and were low cost (Albarbar *et al.,* 2008).

Image removed due to 3rd party copyright.

Figure 2.4: MEMS Capacitive Accelerometer (Sinha et al., 2014)

The piezoresistive accelerometer was also a MEMS device. The piezo resistors generated a change in voltage when subjected to mechanical strain – this strain came from the vibration moving the proof mass on the cantilever. The response – a varying output voltage - was proportional to the vibration measured. The piezoelectric accelerometer produced a charge on its sensing element when subjected to vibration. The piezoresistive and capacitive accelerometers had a DC-response, whereas the piezoelectric accelerometer was AC coupled and was unreliable for velocity or displacement measurements (TE Connectivity, 2017). Piezoelectric MEMS accelerometers also had high output impedances (De Silva, 2015) which caused errors on the signal when connected to low impedances. To overcome the impedance issue, charge amplifiers were placed after the accelerometer on the Integrated Circuit (IC) chip.

Important factors that were considered when choosing the accelerometer included: sensitivity, amplitude range, frequency range and resolution. Sensitivity was the conversion rate of mechanical vibration to voltage output. The amplitude range was the maximum recordable acceleration minus the minimum and when this range was too low signals were saturated and data was lost. The bandwidth was the measurable frequency range of the sensor and the resolution of a digital accelerometer was the smallest acceleration level detectable, it was also often limited by the microcontroller it connected to (Albarbar *et al.,* 2008).

Accelerometers had analog or digital outputs. Figure 2.5 showed a 3-axis IAM-20381 accelerometer.

Image removed due to 3rd party copyright.

Figure 2.5: IAM-20381 Accelerometer (TDKInvensense, 2017, p.1)

This accelerometer had a digital output (*ibid*) because pins 2 and 3 showed “SCL/SPC” and “SDA/SDI” which were part of the digital I2C protocol. The attachment method of the accelerometer to the motor was important. De Silva (2015) stated that glue with appropriate thermal resistance for the location was best because this prevented inadvertently limiting the maximum frequency range of the sensor. As previously mentioned, acceleration was used as the standard unit for vibration. However, velocity was also found to be useful. The velocity was calculated as the integration of acceleration over time. This integration function could have been performed by an electrical integrator *(Brüel & Kjær*,1982) - or by a digital function. The electrical integrator circuit consisted of an op-amp with capacitors and resistors; extra electronic components would have added to the costs and increased manufacturing time. Therefore, a digital solution was decided upon.

The integration of acceleration generated accumulation errors. These accumulated errors also known as drift had to be removed. Numerous methods were investigated to remove and correct drift. One such method was to implement a Kalman Filter. The Kalman filter was an estimator because it estimated the current state of the data by using a weighted average which consisted of the current measured data and the previous predicted set of data. The filter was recursive because it only needed the ‘n-1’th values to generate its next estimated value (Arras and Tipaldi, 2012). Implementation of this filter was dependant on the software and processing capabilities. Drift from the integration of accelerometer data was capable of being removed via a detrending function (SciPy Lectures, no date) which resettled data. Looney, (2014), discussed windowing and filtering. These methods were used to remove unwanted signals and to filter out lower frequencies which helped improve the signal quality after integration. DC offset error was a concern, the error showed as a large amplitude spike at 0Hz. A method identified to remove this was to find the mean of the dataset and subtract it from each data point (Lyons, 2004). Vibration data is periodic which showed that the Fast Fourier Transform (FFT) process was suitable for generating a frequency spectrum. Figure 2.6 showed a spectrum of frequencies from a motor in the press shop which was gathered from the existing expensive VA method.

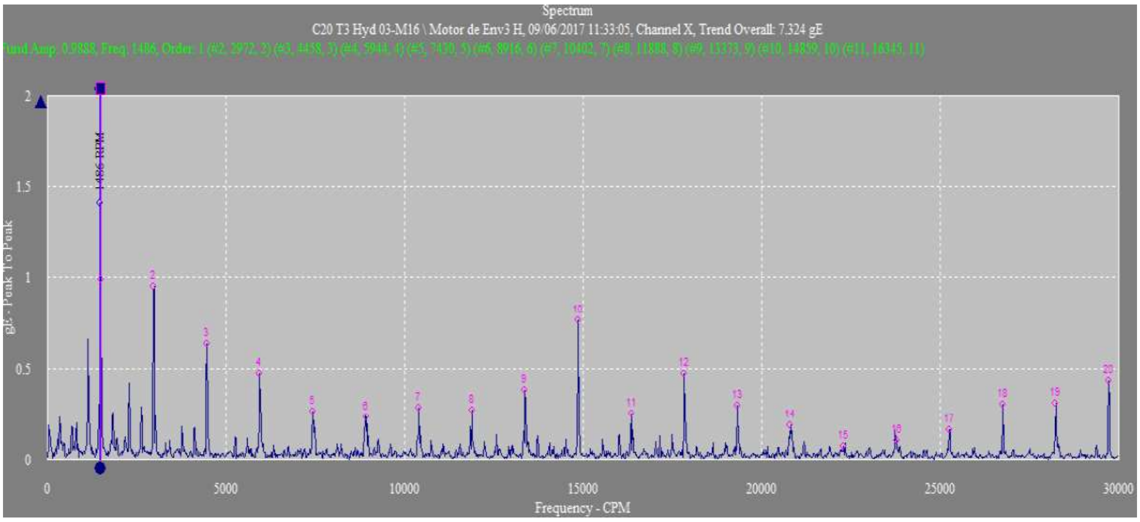


Figure 2.6: Motor Vibration Frequency Spectrum

The highest peak was the fundamental frequency at 24.77Hz which was the running rotations per minute (RPM) of the motor. Subsequent peaks represented harmonics. The data was initially recorded in the time domain but analysis took place in the frequency domain. Orhan *et al.,* (2006), discussed the FFT algorithm as an optimised Discrete Fourier Transform (DFT). The DFT was an application of the Fourier Transform which converted data from the time domain to frequency domain on a discrete data set. The reason the FFT algorithm was used in place of the DFT was because it was faster for datasets with many points. The DFT computation time was proportional to n^2 whereas the FFT was proportional to $n\, log_2\, n$, where n was the number of points (Smith, 2011). The resultant FFT graphs determined the condition of the equipment. Common condition issues were: misalignment, unbalance and mechanical looseness (Mais, 2002).

2.5 Non-Intrusive Pipe Pressure Sensor

2.5.1 Surface Acoustic Wave Method

Surface Acoustic Waves (SAW) were generated by an object and propagated along the parallel axis of it. A SAW sensor consisted of a metallic transducer with intersecting fins on a piezoelectric substrate (Galipeau *et al.,* 1997). Figure 2.7 showed a typical SAW sensor.

Image removed due to 3rd party copyright.

Figure 2.7: SAW Sensor Layout (University of Cambridge, no date)

Different frequency and amplitude SAW were present on the pipes depending on the pressure within. The waves passed through the sensor on the pipe and caused the interdigitated electrodes to move closer together, because they were on a piezoelectric substrate this mechanical movement was converted to an alternating electrical signal (Benetti *et al.,* 2008). The tuning of these sensors was important and because a press had other vibrating elements, misleading SAWs could have been generated and unintentionally measured.

2.5.2 Strain Gauge Method

Sadeghioon *et al*., (2014), discussed a method of using a force sensitive resistor (FSR) to measure the strain on an external pipe wall. They stated that the pressure inside the pipe caused a linear expansion of the pipe radius meaning a force could be detected and a pressure could be derived via the hoop stress strain relationship. Research showed that a measurement of strain was used with other information to derive the internal pressure of an open-ended pipe by using Hoop Stress equations. Strain was a measure of the change in length in comparison to its original length, per Equation 2.4:

$$\varepsilon = \frac{\Delta L}{L} \qquad \text{(Eq. 2.4)}$$

To determine the pressure of the pipes they were noted to be thick-walled. The hoop stress of the pipe varied with its radius because it was a thick-walled pipe (Kadapa, 2017).

The hoop stress for a thick-walled pipe was defined by Equation 2.5: known as Lame's Equation (*ibid*):

$$\sigma_H = \frac{P * r_i^2}{r_o^2 - r_i^2} * \left(\frac{r^2 + r_o^2}{r^2}\right) \qquad (\text{Eq. } 2.5)$$

P was the internal pipe pressure, r_i was the internal radius, r_o was the external radius and r was the radius where the measurement was taking place: the external radius. Equation 2.5 was rewritten as Equation 2.6.

$$\sigma_H = \frac{P * r_i^2}{r_o^2 - r_i^2} * \left(\frac{r_o^2 + r_o^2}{r_o^2}\right) = \frac{2 * P * r_i^2}{r_o^2 - r_i^2} \qquad (\text{Eq. } 2.6)$$

The hoop strain, ε_H, was defined in Equation 2.7 (*ibid*).

$$\varepsilon_H = \frac{\sigma_H - v(\sigma_r + \sigma_l)}{E} \qquad (\text{Eq. } 2.7)$$

σ_H was hoop stress and σ_r was the axial stress which was 0 at the external radius. The steel used in the pipes being measured was E355 – EN1.0580 steel. It had a Poisson Ratio 'v' of 0.29 (Gandy, 2007). Young's Modulus 'E' was the ratio between the stress and strain of the material and σ_l was the longitudinal stress. σ_l was calculated with Equation 2.8:

$$\sigma_l = \frac{P * r_i^2}{r_o^2 - r_i^2} \qquad (\text{Eq. } 2.8)$$

The hoop strain was then calculated from Equation 2.9:

$$\varepsilon_H = \frac{\frac{2 * P * r_i^2}{r_o^2 - r_i^2} - 0.29(0 + \frac{P * r_i^2}{r_o^2 - r_i^2})}{E} = \frac{1.71 * \frac{P * r_i^2}{r_o^2 - r_i^2}}{E} \qquad (\text{Eq. } 2.9)$$

Young's modulus was 200 GPa (Gandy, 2007). Equation 2.10 showed the equation for the internal pressure of a pipe in relation to its hoop strain.

$$P = \frac{E * \varepsilon_H * (r_o^2 - r_i^2)}{1.71 * r_i^2} = \frac{200 \times 10^9 * \varepsilon_H * (r_o^2 - r_i^2)}{1.71 * r_i^2} \qquad (\text{Eq. } 2.10)$$

The strain gauge, seen in Figure 2.8, measured how much an object deformed. There were different types of gauge but they all provided a change in resistance based on the extent of the strain (National Instruments, 1998).

Image removed due to 3rd party copyright.

Figure 2.8: Mounted Metallic Strain Gauge (ibid, p.2)

Other considerations had to be made with resistive based measuring methods. The hydraulic pipes often reached temperatures of 55°C. The resistivity of an object varied with temperature (Boston University Physics, no date) and the coefficient of variation was called the temperature coefficient of resistivity. Equation 2.11 showed this:

$$\rho = \rho_0(1 + \alpha * \Delta T) \qquad (\text{Eq. } 2.11)$$

Resistivity was ρ, original resistivity was ρ_0, α was the temperature coefficient of resistivity and ΔT was the change in temperature. Resistance was proportional to resistivity as per Equation 2.1. An assumption was made that the linear expansion coefficient was significantly smaller than the temperature coefficient of resistivity meaning the resistance changed much more than the length or area (*ibid*). Therefore, it was assumed that the resistance changed with temperature; Equation 2.12:

$$R = R_0(1 + \alpha * \Delta T) \qquad (\text{Eq. } 2.12)$$

R_0 was the initial resistance, α was the temperature coefficient of resistivity and ΔT was the temperature change. It was determined that temperature changes would alter the resistance of the gauges and as such they needed to be accounted for.

Placing a strain gauge in a Wheatstone bridge configuration (Figure 2.9) created an output voltage proportional to the measured strain.

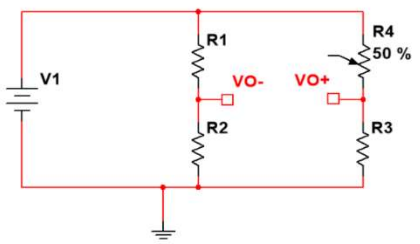


Figure 2.9: Quarter Bridge Circuit

The difference between the positive and negative output terminals were proportional to the difference in resistance between the R_1 R_2 pair and the R_3 and gauge pair. The nominal gauge resistance R4 was herein referred to as R_G and the ΔR element was solely caused by the strain changing. This particular setup was the Quarter Bridge (National Instruments, 1998). The Gauge Factor (GF) of a strain gauge was the ratio of the partial change of resistance compared to the strain measured as expressed by Equation 2.13.

$$GF = \frac{\frac{\Delta R}{R_G}}{\varepsilon_H} \qquad (\text{Eq. } 2.13)$$

The GF for gauges differed depending on the material of construction (*ibid*). ΔR was the change in resistance, R_G was the nominal gauge resistance, GF was gauge factor and ε_H was the hoop strain. Equation 2.13 was rearranged to make the strain the subject and it was substituted into Equation 2.10 which generated Equation 2.14.

$$P = \frac{200 \times 10^9 * \frac{\Delta R}{GF * R_G} * (r_o^2 - r_i^2)}{1.71 * r_i^2} \qquad \text{(Eq. 2.14)}$$

The resistance of the leads from the gauge to the other circuitry was important (HBM, no date) and it was either compensated for in software or hardware. Figure 2.10 showed the setup which considered the lead resistances and included a perpendicular placed dummy gauge.

Image removed due to 3rd party copyright.

Figure 2.10: Strain Gauge Circuit with Thermal Correction (National Instruments, 1998)

The hoop strain was not measured by the dummy strain gauge because it was not positioned on the same axis but the temperature changed the resistance in the same way on both gauges. Therefore, the temperature resistance changes were compensated for (*ibid*).

The varying output voltage of the gauge was not large enough to be accurately monitored by microcontrollers so amplification was necessary. An Instrumentation Amplifier (IA) was commonly used to amplify the voltage of a Wheatstone bridge. The IA was connected to the differential output of the strain gauge and provided high gain and a high Common Mode Rejection Ratio (CMRR). This was important to increase the low voltage differential input to a higher voltage output while minimising common mode interferences which were present on the input signal (Claycomb, 2015). It was also imperative that the power supply to the bridge was stable so that accurate results were transferred between stages.

2.6 Power, Interfacing Circuits and Wireless Transmission

The new multi-sensor system required a stable DC power supply. It was hardwired into existing power systems but circuitry was required to provide the correct voltage levels and current capacity for each sensor. Research was carried out into the best methods for signal conditioning between new and existing systems. The circuits were required to wirelessly transfer some of their data to a device which graphically displayed the results: named the central graphical device (CGD). Various methods for completing this part of the research were also explored.

2.6.1 Power Supply

The Programmable Logic Controller (PLC) control system used a 24V DC power supply. Three popular microcontroller's/microcomputer input power requirements were obtained to determine power requirements. The Arduino MEGA 2560 R3 required an input voltage between 7-12V (Arduino, no date). The schematic diagram was investigated (Guadalupi, 2019). A section was shown in Figure 2.11.

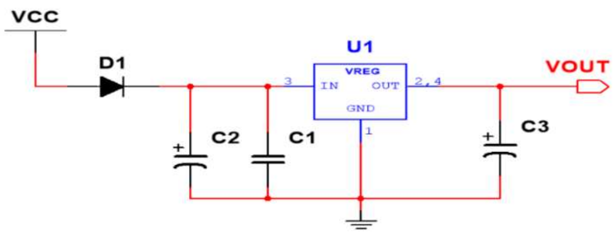


Figure 2.11: Arduino Power Supply Circuit Schematic (ibid)

The datasheet showed that it provided 5V up to 800mA (ST, 2013). Therefore, for an Arduino MEGA 2560 Rev 3 the supply needed to provide at least 7-12V up to 800mA. The Raspberry Pi 3 Model B+ required a 5V DC supply with maximum draw of 2.5A (Raspberry Pi, no date). The Teensy 3.6 required an input voltage of 3.6-6V with a current rating of 250mA (PJRC, no date).

Voltage regulators were deemed suitable for providing power. Linear voltage regulators provided a constant output voltage but had large power dissipation, resulting in the expulsion of heat which needed large heatsinks to alleviate damage (Horowitz and Hill, 2015). Switching regulators used a rapidly switching transistor to charge and discharge inductors and capacitors to maintain a steady output voltage. The output was compared to a reference which allowed for adjustments to be made to the switching frequency for stability. Switching regulators were more efficient and did not need heatsinks (*ibid*). Switching frequency noise was identified as a source of potential interference and was minimised using decoupling capacitors.

2.6.2 Data Signal Conditioning

The PLC output cards provided a 24V DC output when switched 'high' and a 0V DC output when switched low. The signals were not used to drive a load but were control signals which allowed potential dividers to be used to step down voltages. It was assumed the chosen microcontroller/microcomputer would output at 5V or 3.3V DC which was too low for a PLC input card. The in-use Siemens SM321 Input module stated that the required input voltage for a high signal was between 13V and 30V; with '0' being between -30V and 5V (Siemens, 2019). Converting this 5V or 3.3V signal to 24V required additional circuitry. Two options were analysed. One used a boost converter with an alternating duty cycle to boost the output voltage in comparison to its input voltage (Horowitz and Hill, 2015). The second option used optocouplers/optoisolators, one type of optocoupler contained a Light Emitting Diode (LED) and a photodetector, this setup created electrical isolation. The output was controlled by a photodetector which activated depending on the light it received from the LED (*ibid*). This allowed a low voltage signal to switch an LED emitter which switched the photodetector or phototransistor which was connected to a higher voltage for the PLC.

2.6.3 Wireless Transmission Methods

Wireless communication between the sensors and the CGD was crucial for improved CM in factories. The only other devices that transmitted in a wireless capacity were overhead cranes and moving carts; data provided by Plant Swindon showed that 433MHz to 459MHz was occupied and had to be avoided to prevent interference. Three different modules were investigated for their suitability: Table 2.1 showed parameter data of the nRF24L01 transceiver (Nordic Semiconductor, 2006), the ESP8266EX WiFi module (Espressif Systems, 2018) and the HC-05 Bluetooth Module (which used the BLK-MD-BC04-B chip) (ITead Studio, 2010).

Table 2.1: Comparison of Parameters for Wireless Transmission

Parameter	nRF24L01	ESP8266EX	HC-05
Max Output Power	4dBm	20.5dBm	4dBm
Max Current consumption in TX	11.3mA	80mA (average)	19.8mA
Max Current consumption in RX	12.3mA	80 mA (average)	19.8mA
Bandwidth	2.4–2.525GHz	2.4–2.485GHz	2.4–2.485GHz
Channel spacing Max Data Rate	2MHz	5MHz	2MHz
Max Data Rate over air	2Mbps	72.2Mbps	3Mbps
Digital Interfaces	Serial Peripheral Interface (SPI)	SPI, I2C	SPI
Input Voltage Level Max	5V	3.6V	3.6V
Supply Voltage Range	1.9V to 3.6V	2.5V to 3.6V	3.6V to 5V

The current consumption readings for the HC-05 module were from a different source: (Bolutek, no date). The Channel Spacing at Max Data Rate value for the HC-05 module was from (Bluetooth, 2019). Analysis of the table showed clear benefits and disadvantages of each module. The nRF24L01 module was a simple Radio Frequency (RF) transceiver with low current consumption – the lowest of the three – but with a reasonable power output of up to 4dBm. All 3 devices shared the 2.4GHz Industrial, Scientific and Medical Radio Band (ISM) avoiding the already reserved range. All devices utilised SPI to communicate between the microcontroller and the transceiver. The ESP WiFi module had a higher maximum data rate at the cost of higher power requirements but all devices operated on similar voltage supplies. The nRF module was 5V tolerant for inputs which meant it could handle 5V and 3.3V based I/O. The HC-05 used Bluetooth protocol class 2 making it a point-to-point device (*ibid*) and not suitable for the system. The nRF24L01 utilised a star network configuration (Nordic Semiconductor, 2006) which ensured multiple devices could have been connected to the CGD with correct addressing and it had sufficient range.

2.6.4 Central Graphical Device

The CGD was designed to receive data from the sensors and have it displayed on a screen via a graphical user interface (GUI). Devices such as tablet computers were explored but were expensive, therefore the Raspberry Pi and a Pi touchscreen were investigated. The device was required to be portable so it needed a portable battery pack capable of supplying both the microcomputer and screen. The Raspberry Pi contained Python and within that was a module called appJar (appJar, 2018) for GUI design. The Raspberry Pi also contained General-Purpose Input Output (GPIO) pins for SPI communications where required (RaspberryPi, no date (b)).

2.6.5 Microcontrollers

Table 2.2 showed important characteristics of 3 different microcontrollers/microcomputer: Arduino MEGA 2560 (Arduino, no date); Teensy 3.6 (Hobbytronics, no date) and the Raspberry Pi Model 3 B+ (ThePiHut, no date).

Table 2.2: Microcontroller Specification Table

Feature	Arduino MEGA	Teensy 3.6	Raspberry Pi 3B+
Microcontroller/Processor Chip	ATmega2560	MK66FX1M0VMD18	BCM2837
Input Voltage	7-12V	5V	5V
Digital I/O Pins	54	45	40 GPIO
Digital I/O Max Voltage	5V	3.3V	3.3V
Clock Speed	16MHz	180MHz	1.4GHz
Programming Environment	Arduino IDE	Teensyduino IDE	Python
Interfaces	I2C, SPI	I2C, SPI	I2C, SPI

All had a high number of I/O pins. The clock speed of the Raspberry Pi was the best because it was a microcomputer frequently used for high performance tasks. The speed of the Teensy was much greater than that of the Arduino. The Raspberry Pi supported programming in Python which could perform complex mathematical functions such as FFT. All devices were capable of utilising the common peripheral interfaces of I2C and SPI. In summary, the software and hardware of the Arduino was most familiar and it had the highest number of I/O. The speed of the Raspberry Pi was superior. The Teensy was suitable but unfamiliar and not worth the extra cost over the Arduino. The Arduino and Raspberry Pi appeared most suitable.

2.7 Practical Limitations and Interferences

Potential wireless interferences were minimised by choosing an unused base operating frequency for transmission. Generally, regulators had two decoupling capacitors – one at the input and one at the output which shunted the AC signals down to the capacitor allowing the DC signals to continue through. This setup also minimised the effect of voltage spikes and high frequency noise interference which were known to be common in switching regulators. Electrical noise was a common mode of electrical interference and was prevalent in the factory environment. Sensors using lower voltage levels were more susceptible to this distortion and could have inadvertently misrepresented the actual signal. Electromagnetic Interference (EMI) was caused by high power AC lines, whereby a magnetic field was generated due to the AC current through the conductor. If there were nearby unshielded low voltage cables then the magnetic field could have been inducted creating EMI (Horowitz and Hill, 2015). Research showed some steps taken to minimise EMI included: routing the signal cables away from the high-power cables and utilising non-conductive communication methods such as fibre optics (De Silva, 2015).

Noise was generated when circuits were improperly grounded – Ground Loop Noise (GLN). GLN occurred when two devices connected to grounds at different physical points had different potentials. If the potential at one ground point was higher than the other grounded point then there was a potential difference. GLN was removed by using an insulated junction box and having only one ground point (*ibid*). Isolation transformers or other electrically isolated components were also used to isolate the signals between two devices because this isolation broke the ground loop (Whitlock, 2008). Ingress protection (IP) rated enclosures prevented interference from external materials.

2.8 Summary

A thorough literature review was undertaken, with different author’s views and information being compared and contrasted. The information obtained throughout this literature research enabled the initial designs and specifications to come to fruition in the next chapter.

3 Specifications and Designs

3.1 Introduction 3-21
3.2 Common Requirements 3-21
3.3 Clutch and Brake Sensor 3-22
3.4 Vibration Analysis Sensor 3-27
3.5 Non-Intrusive Pipe Pressure Sensor 3-33
3.6 Power, Interfacing Circuits and Wireless Transmission 3-44
3.7 Programmable Logic Controller Programming 3-50
3.8 Summary 3-62

3.1 Introduction

This chapter covered all of the designs for the sensor circuits, power circuits and interfacing circuits. The common requirements of the entire system were identified and the wireless transmission designs and PLC programming were also derived.

3.2 Common Requirements

There were common requirements applicable to all 3 sensors which were written up into Table 3.1.

Table 3.1: Common Requirements for all Circuits

Requirement	Reasoning
High IP rated enclosure	Protected from the press shop elements. Proper grounding.
Two-position switch to turn power on and off to the sensor circuits.	Safety purposes.
Power indication light on enclosure	Safety purposes.
Master Control Bit in the PLC program.	Easy access bypass.
Miniature Circuit Breaker (MCB) on the 24V in line.	Protected the PLC from shorts or other electrical issues with the sensors.
A Lifebit sent continuously between the Microcontroller and PLC.	Both devices knew if the other was on and working. Prevented data recording if no lifebit received or transmitted.
Upper and Lower limits set on each adjustable parameter.	Protected the press from mechanical damage.

Including these requirements for the entire system helped to ensure the system was safe to use.

3.3 Clutch and Brake Sensor

The C&B sensor initial designs and specifications were based upon the IR sensor explored in the literature. The infrared sensor was mounted opposite a white reflective object and connected to the microcontroller analog input. The sensor measurement was calibrated when the sensor was initialised and the pad wear was noted as a difference from this datum point.

The measurement was designed to be taken when the brake was engaged with the press at TDC. The pad being measured when new was 1.75” (44.45mm) thick. As time progressed the thickness of this pad decreased. The previous manual method used a dipstick inserted between the brake pad and the opposite plate to determine the gap present when the brake was disengaged. Table 3.2 showed the expected gap measurements:

Table 3.2: Expected Measurements - Manual method

State of measurement	Inches	Millimetres
Nominal Gap	1/8	3.175
Biggest Gap	1/4	6.350
Smallest Gap	1/32	0.794

Due to wear the gap deviated from the nominal to the bigger gap. Data from the manual method was categorised into: OK, Wearing and NOK (Not OK). Table 3.3 showed the ranges of these categories from the nominal to the biggest gap range.

Table 3.3: Ranges and Categories of Wearing Pads

Range (inches)	Category
1/8 → 5/32	OK – Green
11/64 → 13/64	Wearing – Amber
7/32 →1/4	NOK - Red

Metric units were used moving forward, the 1/8” was set as the 0 datum. The width of each category was calculated as determined by Table 3.4.

Table 3.4: Metric Ranges and Categories of Wearing Pads

Range (mm)	Width (mm)	Category
0 → 0.794	0.794	OK – Green
0.794 → 1.588	0.794	Wearing – Amber
1.588 → 2.382	0.794	NOK - Red

The 0mm starting point referenced the fact the sensor was calibrated to a zero position when new pads were fitted with a gap of 1/8” or 3.175mm. Each category spanned 0.8mm and so the chosen sensor needed to have a better resolution than 0.8mm or 0.08cm. The Sharp GP2Y0A41SK0F had a 4cm to 30cm range with good linearity between 6-20cm - and the reflectance ratio of the object was not strictly important (Sharp, no date). Figure 3.1 showed the analog output voltage of the sensor against the inverse of the measured distance.

Image removed due to 3rd party copyright.

Figure 3.1: Output Voltage Graph for the SHARP GP2Y0A41SK0F (Sharp, no date, p.5)

The program was written to use centimetres. Between 6-7cm there was a voltage difference of 0.27V. The Arduino MEGA 2560 Analog to Digital Converter (ADC) had a 10-bit resolution (Arduino, no date). The number of distinguishable levels was 2^{10} which was 1024 levels. The range of the ADC was between 0-5V making the resolution for an analog input: $\frac{5}{1024} = 0.005V$. The number of levels in the 6-7cm range was: $\frac{0.27V}{0.005V} = 54$. Converting this back into a resolution of the distance measurement found that each level represented: $\frac{1cm}{54} = 0.019cm$; rounded up was 0.02cm. The required smallest measured range was 0.08cm which meant that 4 different measurements were capable of being made in each range with this sensor. It was deemed suitable.

An equation which determined the distance from the input voltage was necessary. The linearity of Figure 3.1 from 6.5cm to the origin showed a standard $y = mx + c$ relationship extended to the origin. Over this range the change in the y-value was 1.88V and the change in the x-value was 0.143. The gradient 'm' of this line was $\frac{1.88}{0.143} = 13.147$ and the y-intercept 'c' was 0. The equation was: $y = 13.147x$, where $x = \frac{1}{s+0.42}$ with 's' as distance. The combined linear equation was shown in Equation 3.1:

$$y = \frac{13.147}{s + 0.42} \qquad \text{(Eq. 3.1)}$$

Equation 3.2 showed the rearrangement for s:

$$s = \frac{13.147}{y} - 0.42 \qquad \text{(Eq. 3.2)}$$

'S' was distance and 'y' was the output voltage. The equation was noted to have limited accuracy due to the nature of its generation, however it was easily adjustable. The analog read function of the microcontroller used levels which required conversion to a voltage. With 1024 levels between 0-5V each level represented 0.0048828125V, or 4.883mV. Equation 3.2 was modified so that 'y' was converted from a level to a voltage: showed in Equation 3.3.

$$s = \frac{13.147}{(l_i * 0.0048828125)} - 0.42 \qquad \text{(Eq. 3.3)}$$

's' was distance, y was replaced by '$l_i * 0.0048828125$' where 'l_i' was the level the microcontroller's ADC read from the sensor.

The circuit contained 5 parts: Power Supply Regulation, Microcontroller to PLC Interfacing Circuits, PLC to Microcontroller Interfacing Circuits, Calibration Button and the Analog Measurement Sensor. The power supply regulation was a switching regulator that generated 9V (up to 1A) with a 24V input taken from the PLC. The microcontroller to PLC interfacing circuits were designed as a series of opto-couplers and resistors, more detail was provided in section 3.6.3. The PLC to Microcontroller circuit used a potential divider to decrease the voltage from 24V to 5V. The calibration button was connected through a resistor to the microcontroller. The analog measurement sensor had 3 wires, 2 for power (took from the microcontroller) and the remaining wire was the voltage signal. Figure 3.2 showed the block design of the system.

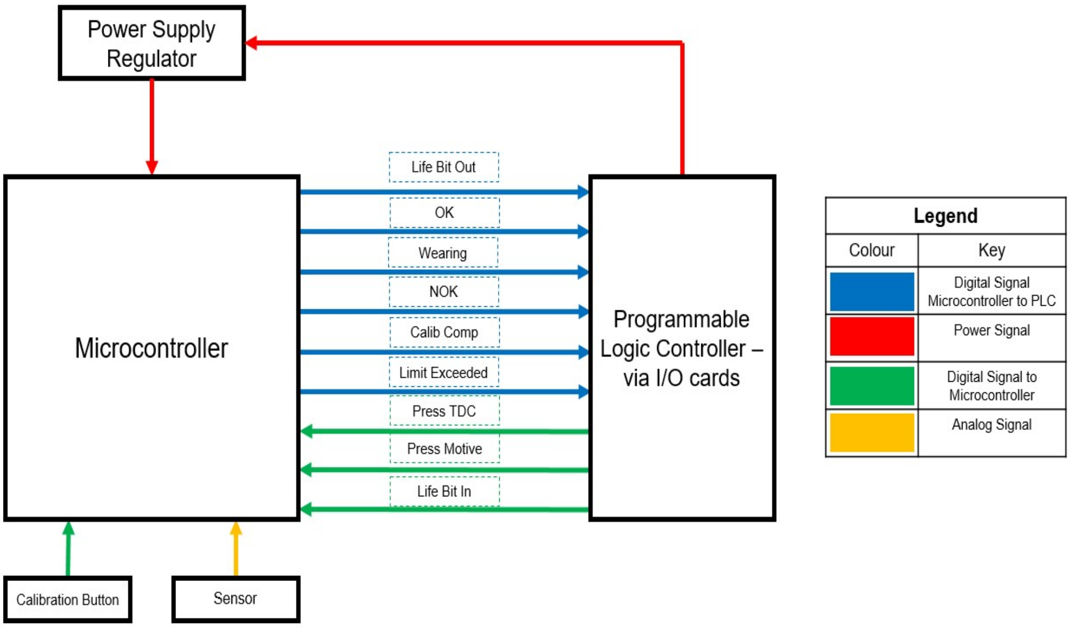


Figure 3.2: Block Diagram for Clutch and Brake Sensor

On the basis of this design the initial program was written and added to Appendix A1. The first part of the code setup the pin mapping and internal variables. Each pin was given a label which was referenced throughout. The float data type was used for the internal variables that held the distance measurement to ensure accurate values were captured. After these variables and pins were defined the setup routine was called once on initialisation and it set the digital pin variables as either an input or output. The calibcomp internal variable was set low to ensure that a calibration was performed. Serial communications were activated as they were used in testing for monitoring of variables via the Integrated Development Environment (IDE) serial monitor.

The main loop of the program performed the measurements and calculations. A series of condition checks were used. The first statement checked that the PLC sent the lifebit to the microcontroller, if it was detected then a lifebit signal was sent to the PLC. A second conditional check checked if the Press TDC signal was high and the Press Motive signal was low; these conditions had to be met before a measurement was taken. Then the sensor needed calibration. When the calibration button was pressed the distance was read and stored in the variable called ‘measurement’. The conversion to voltage was completed and stored in ‘measasvolt’. The calibration distance was then calculated using the equations and stored in ‘calibdistance’. At that point ‘calibcomp’ and ‘calibcompinternal’ were set high.

The actual measurements were now taken. The same process of reading the sensor value and converting it to a voltage was completed, but a difference was required instead of a distance. Therefore, the newly measured distance was subtracted from the calibration distance and stored in ‘difference[i]’ where i ranged from 0-9. The difference was always positive because as the pad wore the distance got smaller and the calibration distance was always largest.

A ‘for loop’ took 10 measurements and stored them in separate variables. Once the 10 measurements were taken the average difference calculation was completed and stored in ‘avgdifference’. The other iterative for loop rotated through ‘j’ from 0-2 which generated 3 averages of 10 measurements. The average after ten readings was transferred from ‘avgdifference’ to ‘avgone’, ‘avgtwo’ and ‘avgthree’ respectively. The process was quick to ensure all measurements were taken before the press cycled.

The averages were then checked against a set of CM boundaries. To ensure only correct results were transmitted – the 3 sets of averages taken needed to be in the same range. If all of the average differences were ≥0 and ≤0.08 then the OK signal was activated; if they were >0.08 and ≤ 0.16 then the Wearing signal was activated; if they were >0.16 and ≤1.00 then the NOK signal was activated. If all averages were <0 then the error signal was sent to the CGD. If all averages were >1.00 then the Limited Exceeded signal was activated.

The circuit schematic for the C&B interfacing and calibration circuitry was added to Appendix A2. The schematic showed the calibration button, 3 potential dividers for converting PLC signals and 6 potential dividers to reduce the microcontroller signals for use with the optocouplers.

3.4 Vibration Analysis Sensor

The VA sensor was based on a capacitive MEMS accelerometer with a digital output because the signal needed to be integrated to obtain velocity and capacitive MEMS accelerometers were also generally small and cheap. The vibrations were measured in one axis. The maximum acceleration to be measured and the highest frequency was identified from existing VA data; it was determined that the frequency range was 0-1kHz and the maximum acceleration was ±10g.

The units of measurement for vibration were g and mm/s – acceleration and velocity. The root mean square (RMS) of velocity was also frequently used in VA; the maximum RMS value of velocity was equivalent to 0.707 times the peak (Hansford Sensors, 2020). The ADXL345 accelerometer was chosen because: the frequencies required were <5kHz, it was a capacitive MEMS accelerometer (Varanis *et al.,* 2018) and was widely available. The datasheet for the ADXL345 provided information on its capabilities (Analog Devices, 2015). The acceleration range was ±16g with a 13-bit resolution at this maximum range. The supply voltage was between 2-3.6V. The accelerometer used both I2C and SPI data interfaces. The temperature range was up to 85°C – which allowed direct mounting. The ADXL345 was capable of 3-axes measurement but only 1 was required; however, all 3-axes had readings taken of them in case they were needed. The sensitivity change due to temperature was ±0.01%/°C and due to the low temperatures encountered this resulted in little difference in sensitivity. The ADXL345 had been factory calibrated but the 0g offsets were checked upon initial readings at rest and the offset was removed in the program.

One of the most important parameters of the ADXL345 was its output data rate (ODR): the maximum ODR was 3200Hz with SPI and not I2C because it was limited to a lower rate (*ibid*). To ensure important data was not lost during sampling the Nyquist Sampling theorem was adhered to. The theorem stated that the sampling rate (ODR) was at least 2 times the highest frequency recorded (Berkeley, no date). Therefore, the sampling rate of this accelerometer needed to be at least 2000Hz and the ODR was set to 3200Hz which ensured there were no sampling rate issues. The Raspberry Pi controller used the SPI protocol at 2MHz which enabled sufficient processing speed.

Figure 3.3 contained the block diagram of the accelerometer. Initially, the acceleration signal was ‘sensed’ and prepared for analog to digital conversion. The resultant output from the ADC was then filtered – the data could be stored in a first in first out (FIFO) buffer - and then data was transmitted through serial I/O via SPI or I2C.

Image removed due to 3rd party copyright.

Figure 3.3: Block Diagram of ADXL345 Accelerometer (Analog Devices, 2015)

The mechanical structure of the accelerometer was built upon a polysilicon surface. Polysilicon springs held the moving structure and vibration caused deflection of the structure. The fixed plates connected to the structure and moving surface created a varying capacitance which was proportional to the current acceleration (*ibid*). The noise was minimised with digital filtering. A higher supply voltage was known to decrease noise on the signal and so 3.3V was used (*ibid)*.

The accelerometer was connected to the Raspberry Pi 3B+ controller for data capture and retrieval. Nagimov (2006), created a program under a copyleft GNU General Public License to retrieve the data from the ADXL345 and store it in a .csv file - Appendix A3. The program interacted with the ADXL345 registers to harvest data. The program was thoroughly analysed to guarantee it met the requirements. The CM setup required minimal manual interference and so on main start-up a line of code was entered once into the Raspberry Pi terminal:

```
watch -n 600 sudo ./adxl345spi -f 3200 -t 2 -s data.csv
```

The Pi recorded 2 seconds of data, sampled at 3200Hz, every 600 seconds to data.csv and the process repeated every 10 minutes. The ADXL345SPI.c program written by Nagimov began by declaring the modules used. It then defined variable declarations which corresponded to initial values of different registers in the ADXL345.

```
#define DATA_FORMAT   0x31
#define DATA_FORMAT_B 0x0B
#define READ_BIT      0x80
#define MULTI_BIT     0x40
#define BW_RATE       0x2C
#define POWER_CTL     0x2D
#define DATAX0        0x32
```

The data format register 0x31 (x represented hexadecimal) was used to determine how the data was presented in the following data registers. The register was 8-bits long (Analog Device, 2015) and contained the following functions as per Table 3.5.

Table 3.5: ADXL345 0x31 Register (ibid)

D7	D6	D5	D4	D3	D2	D1	D0
SELF_TEST	SPI	INT_INVERT	0	FULL_RES	Justify	Range	

Register 0x31 held the value 0x0B (00001011 in binary) setting the range correctly and providing the full-resolution of 13-bits. D0 and D1 determined the range of the accelerometer and were set to ‘1’ for ±16g. ‘Self_test’ was 0 as it was unused; ‘SPI’ was 0 because 4-wire SPI was used instead of 3-wire; ‘Int_Invert’ was 0 so that interrupts would have been active high and D4 was 0. The ‘justify’ function switched between most and least significant bits and was not required.

The variables ‘read_bit’ and ‘multi_bit’ related to SPI reading functions. The BW_RATE register 0x2C was set to 0x0F (00001111 in binary); low power was not used and the ODR was set to 3200Hz. The power control register 0x2D determined how the accelerometer measured. Table 3.6 showed this register.

Table 3.6: ADXL345 0x2D Power Control Register (ibid)

D7	D6	D5	D4	D3	D2	D1	D0
0	0	Link	Auto_Sleep	Measure	Sleep	Wakeup	

The link function was intended for activity detection measurements. The auto-sleep function was useful for energy conscious applications. The D3 bit when ‘1’ put the accelerometer into measurement mode. The sleep bit was set to 0 and the wakeup bits were unnecessary as they were only used if sleep mode was active. In the program, this register was set to 0x08 (in binary 00001000) so that only continuous measurement mode was engaged. The DATAX0 register 0x32 was the first register (they went up to 0x37) that stored data. Next, the ‘freqMax’ variable was set to 3200Hz and the ‘speedSPI’ variable was set to 2MHz. The program output the acceleration in the unit ‘g’. The program ignored the first few samples upon start-up to remove errors and then read the data for each axis and stored it in the .csv file.

The FFT was performed in a program written in Python and added to Appendix A4. The program initialised the modules used for calculation of the FFT, plotting graphs, filtering, integration and other functions. Several resources were used throughout the process. The scientific and mathematical functions utilised the SciPy module and resources (SciPy Cookbook, 2015; SciPy, 2019). The NumPy module was used for numerical purposes with guidance from (NumPy, 2019). Taking data from the .csv file and using it in Python required the Pandas module (Pandas, 2020). Finally, the data needed to be displayed on graphs so MatPlotLib was used (MatPlotLib, 2020).

The nRF test program was imported and the serial line was allocated to the USB connection with the input flushed. Then the GPIO was setup to send and receive signals from the PLC. The main program was enveloped in an infinite 'while' loop. A sleep function was added at the end of the program to pause it for a set period. After the loop began there was a check to ensure that the 'Lifebit from PLC' and 'Press at bottom' signals were high and low respectively; then the 'Lifebit to PLC' output was set high, it was turned off at the end of the program and set again if the conditions were met. The next stage of the program read data from the .csv file. The data was stored in the 'path' variable and the columns taken from the csv file were corrected to remove the white space before the character. Then the individual columns were stored in their own variables. The DC offset was removed from the data by subtracting the mean of the data from each data point. The number of samples were counted - if the FFT algorithm was required to be run in a time-critical operation then the number of samples would have to have been a number from the power of two (Smith, 2011). The sample rate and time between samples were calculated. The next stage was the velocity calculation. First, the DC-offset removed data variable 'x' was transferred into a new array for processing and the acceleration data was converted to m/s/s in array 'x_conv'. The integration method most suited was the cumulative trapezium rule which was a built-in module function. Once the velocity was calculated, it was converted to mm/s. The RMS values of the mm/s velocity were also calculated and stored in a variable. The RMS calculation multiplied each amplitude data point by 0.707 to reduce the amplitudes of the entire spectral line (Siemens Simcenter, 2019).

The first plot was of the acceleration data with DC offset removed against time. All figures were saved with a file name and date/time stamp on them. The second plot generated was the time domain-based velocity data. The velocity signal showed a clear linear drift caused by integrating the acceleration signal. The method for removing this drift was to perform a linear detrend and then apply a high pass filter. The SciPy package had a linear detrend function and it was applied to both the mm/s and mm/s RMS signal. Both of these signals were then filtered via a 10^{th} order digital high pass Butterworth filter. The removal of the drift and filtering out low frequency signals enabled clearer analysis of the velocity data; graphs were plotted showing the cleaned mm/s and mm/s RMS signals.

The next stage performed the FFT on the acceleration and mm/s RMS signal. The first stage generated the x-axis of the FFT graph in the frequency domain. The plot was required to have evenly spaced numbers over the sample range, the signal started at 0Hz and therefore the end point was calculated by 1.0/(2.0*T) where T was the time between samples. The 'N/2' declaration ensured they were evenly spaced. The next stage of the FFT process was to perform the FFT algorithm on the y-axis data. Only one line of code was required to perform the FFT function, seen in Equation 3.4:

$$yf = fft(x) \qquad \text{(Eq. 3.4)}$$

Where yf was the FFT of the x dataset. The data needed normalising from the FFT operation to be plotted on the graph. Equation 3.5 was used:

$$yff \ = \ 2.0/N \ * np.abs(yf[0{:}\,np.int(N/2)]) \qquad \text{(Eq. 3.5)}$$

Where N was the sample number and yf was the FFT of the x dataset within the specified range. This ensured that only the positive real data was plotted for analysis. The FFT graphs were then plotted and the x-axis had an adjustable range for more detailed inspections. An important part of the program was the ability to categorise the condition of the monitored equipment as either OK, Wearing or NOK. Conditions were setup for mechanical looseness, misalignment, unbalance and good running condition.

Analysis of the FFT peaks was necessary and a program was used which was created by Endolith (2011) in the public domain of free to use code. This code was incorporated into the FFT program. Two arrays of maximum and minimum peaks were identified and processed through the 'peakdet' function. The function needed a 'v' declared which was the yff FFT signal, the 'x' which was the xf of the same FFT graph and a delta value. The delta value determined whether or not a point was a maximum, essentially it was a prominence marker. A peak was considered to be a maximum if the data point before it was lower by 'delta' or more. The identified peaks had their frequency and acceleration/velocity points coupled together and sorted. These peaks were used in conditional if statements where information was printed to the Shell and the serial write function was activated sending the data to the Arduino for decoding while the correct GPIO was powered on. For misalignment according to (Mais, 2002) the key factor was when the 2nd harmonic was between 50%-150% of the amplitude of the 1st harmonic (fundamental frequency) – this was considered wearing. If the 2nd harmonic exceeded 150% amplitude of the 1st harmonic then it was reported as NOK and repaired as soon as possible. Unbalance was characterised by a significantly higher amplitude on the 1st harmonic than usual and was considered as wearing. Mechanical looseness was determined by a series of peaks at more than 3 harmonics whose amplitude was at least 20% of the 1st harmonics peak amplitude. The amplitudes for these CM measurements were identified as being either velocity 'mm/s RMS' or acceleration 'g' so either set of data could be used in condition analysis (SKF, 2000). Figure 3.4 showed the block diagram design of the VA sensor system.

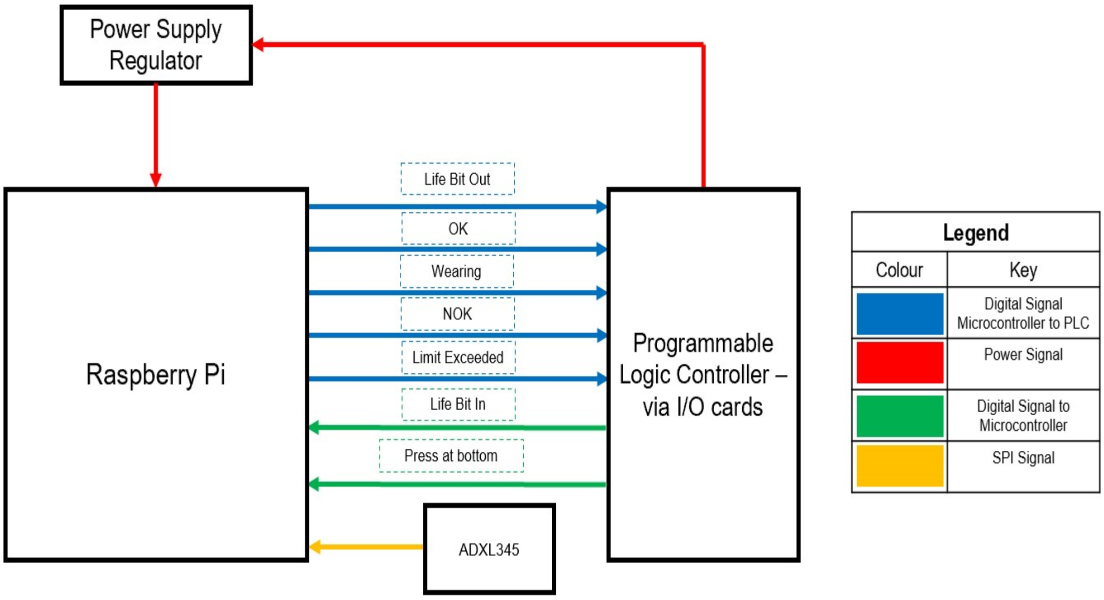


Figure 3.4: VA Sensor System Block Diagram

The block diagram illustrated the connections between the PLC and the Raspberry Pi's GPIO. A 'Press at bottom' signal was designed to act as a prohibitor for the VA program to run; if the press was near its bottom position then the VA was not performed to reduce interference from external vibration. The VA sensor was designed to use the Raspberry Pi 3B+ controller which needed a 5V supply with 2.5A. A different component supplied this sensor compared to the Arduino controllers due to its higher power requirements. A 7" touchscreen was connected to the Raspberry and took its power from the GPIO pins. The JTD20 Series DC-DC Converter was used because it was an isolated converter with up to 1600V DC isolation and up to 4A for a 5V output voltage (XP Power, 2017). Appendix A5 contained the electronic circuit schematic for the interfacing circuitry for the VA sensor. The circuit showed the DC-DC converter, 3 potential dividers for converting PLC signals to Raspberry Pi GPIO signals, potential dividers for the optocouplers which differed slightly because of the lower voltage into the divider and the remainder of the optocoupler system.

3.5 Non-Intrusive Pipe Pressure Sensor

The NIPP sensor utilised a strain-based system. Two BF350-3AA metallic strain gauges (HobbyComponents, no date), were mounted using a high capacity thermal glue in perpendicular orientations to account for temperature variation as part of a Wheatstone bridge. The adhesive and heatshrink insulation helped electrically insulate the gauge from the metallic pipe. The two strain gauges had the same nominal resistance and so did the two other resistors of the bridge. Changes in the strain caused by the internal pressure in the pipe was measured as a change in voltage. This voltage change was small and required amplification by an IA. The IA helped reduce interference on the signal due to its good CMRR. The amplified signal was read by the microcontroller and the internally derived equation converted it to a pressure which was placed into a bracket. The bracket information was then transmitted to the PLC which gave commands to cycle stop the press and shut off the pumps if it detected an unexpected and sustained drop in pressure.

Information of the monitored pipe and the strain gauge specifications were collated in Table 3.7.

Table 3.7: Known Values Table

Known Values Table		
Variable		**Values**
Internal Radius, r_i	(mm)	25.40
Internal Radius Squared, r_i^2	(mm^2)	645.16
External Radius, r_o	(mm)	38.10
External Radius Squared, r_o^2	(mm^2)	1451.61
Young's Modulus, E	(GPa)	200.00
Gauge Factor, GF		2.10
Nominal Resistance, R_G	(Ω)	350.00
Input Voltage to Bridge, V_{IN}	(V)	5.00

The pressure ranged between 0-100bar. Using the data from Table 3.7 and by checking pressures from 0-100bar (converted to Pascals) the theoretical strain from the gauge was calculated with Equation 2.9.

$$\varepsilon_H = \frac{1.71 * \frac{P * r_i^2}{r_o^2 - r_i^2}}{E} = \frac{1.71 * P * r_i^2}{E * (r_o^2 - r_i^2)} \qquad \text{(Eq 2.9)}$$

The results from the calculations were collected in Table 3.8.

Table 3.8: Strain Calculations with 5bar intervals of pressure

Pressure (bar)	**Pressure (MPa)**	$\mu\varepsilon_H$
0	0.00	0.00
5	0.50	3.42
10	1.00	6.84
15	1.50	10.26
20	2.00	13.68
25	2.50	17.10
30	3.00	20.52
35	3.50	23.94
40	4.00	27.36
45	4.50	30.78
50	5.00	34.20
55	5.50	37.62
60	6.00	41.04
65	6.50	44.46
70	7.00	47.88
75	7.50	51.30
80	8.00	54.72
85	8.50	58.14
90	9.00	61.56
95	9.50	64.98
100	10.00	68.40

As expected, the strain values were very small and the gauge limit was 2% (*ibid*) which meant the measurable strain was limited to 0.02 – the calculations showed this would not be exceeded. Equation 2.13 was rearranged to make ΔR the subject.

$$\Delta R = R_G * GF * \varepsilon_H \qquad \text{(Eq. 2.13)}$$

Table 3.8 expanded to include the resistance change to make Table 3.9.

.

Table 3.9: Pressure, Strain and Resistance Changes

Pressure (bar)	Pressure (MPa)	$\mu\varepsilon_H$	ΔResistance $(m\Omega)$
0	0.00	0.00	0.000
5	0.50	3.42	2.514
10	1.00	6.84	5.027
15	1.50	10.26	7.541
20	2.00	13.68	10.055
25	2.50	17.1	12.569
30	3.00	20.52	15.082
35	3.50	23.94	17.596
40	4.00	27.36	20.110
45	4.50	30.78	22.623
50	5.00	34.2	25.137
55	5.50	37.62	27.651
60	6.00	41.04	30.164
65	6.50	44.46	32.678
70	7.00	47.88	35.192
75	7.50	51.3	37.706
80	8.00	54.72	40.219
85	8.50	58.14	42.733
90	9.00	61.56	45.247
95	9.50	64.98	47.760
100	10.00	68.4	50.274

The total range of ε_H and ΔR – excluding 0bar – was added to Table 3.10.

Table 3.10: Ranges of Strain and Resistance

Ranges	
Range of ε_H	6.50E-05
Range of ΔR	0.0477603

The resistance changes were split into different groups of pressure – initially 50bar, then 10bar and then 50bar again. Table 3.11 highlighted these.

Table 3.11: Difference in Resistance between Pressures

ΔResistance $(m\Omega)$ between Pressures	
0bar to 50bar	25.137
50bar to 60bar	5.027
50bar to 100bar	25.137

The 0bar to 50bar range and the 50bar to 100bar range were both the same and as expected the resistance increased with pressure. The 50bar to 60bar range was very small and showed the need for high levels of amplification. The calculations showed that when there was no pressure there was no strain present – in reality this differed hence the need for calibration.

The remaining 3 resistors including the dummy gauge were modelled as resistors with a set resistance for the theoretical calculations. The nominal value of all these resistors was idealised as 350Ω. However, 3 sets of tolerances were calculated and applied to give a broad range of testing scenarios; Table 3.12 showed these.

Table 3.12: Resistors Resistance Tolerances

Resistors (Ω)			
Nominal	350		
±1% Tolerance	346.5	to	353.5
±0.1% Tolerance	349.65	to	350.35
±0.01% Tolerance	349.965	to	350.035
Deviation from nominal (Ω)			
±1% Tolerance	3.5	to	3.5
±0.1% Tolerance	0.35	to	0.35
±0.01% Tolerance	0.035	to	0.035

The next stage of calculations involved modelling the Wheatstone bridge with these calculated resistance values. ΔR was added to the nominal value of the strain gauge which was 350Ω. Table 3.13 summarised all of the calculations performed from 0-100bar in 10bar increments. R1 was designated as the active strain gauge, R2 was the dummy gauge and its resistance was changed from nominal to the highest and lowest of the 1% tolerance; this was done to reflect the likelihood of encountering larger tolerances in practicality. The output voltage of this potential divider was then calculated; the output of the other divider in the bridge is labelled as P1. These fixed value resistors were set to smaller tolerances of 0.01% because laser engraved precision 0.01% tolerance resistors were purchased. The voltage output of this divider (P1) assumed one resistor on the higher end of tolerance and one of the lower; these resistors were located in the same location to allow for temperature to affect them both equally. The change in the output voltage was P2 minus P1. Many assumptions had to be made throughout to obtain the final equation which could have led to differences in the results from what was initially expected. The next stage of the sensor circuit was for the voltage to be amplified with an IA. Three different gain levels were used in calculations: 10, 100 and 1000.

Table 3.13: Sensor Calculations Overview

Output voltage calculations from a series of pressures and calculated hoop strains																
		Strain Gauge Potential Divider														
		Pressure	R1	R2	Vin	Vout	Vout P1	Vout P2	ΔVout	Gain	IA Vout	Gain	IA Vout	Gain	IA Vout	
	Nominal	0	350	350	5	2.5	2.49975	2.5	0.00025	10	0.0025	100	0.025	1000	0.25	
0 bar	**Lower**	0	350	346.5	5	2.487437	2.49975	2.487437	-0.01231	10	-0.12313	100	-1.23128	1000	-12.3128	
	Higher	0	350	353.5	5	2.512438	2.49975	2.512438	0.012688	10	0.126878	100	1.268781	1000	12.68781	
	Nominal	1000000	350.005	350	5	2.499982	2.49975	2.499982	0.000232	10	0.00232	100	0.023205	1000	0.232045	
10 bar	**Lower**	1000000	350.005	346.5	5	2.487419	2.49975	2.487419	-0.01233	10	-0.12331	100	-1.23308	1000	-12.3308	
	Higher	1000000	350.005	353.5	5	2.51242	2.49975	2.51242	0.01267	10	0.126699	100	1.266986	1000	12.66986	
	Nominal	2000000	350.0101	350	5	2.499964	2.49975	2.499964	0.000214	10	0.002141	100	0.021409	1000	0.214091	
20 bar	**Lower**	2000000	350.0101	346.5	5	2.487401	2.49975	2.487401	-0.01235	10	-0.12349	100	-1.23487	1000	-12.3487	
	Higher	2000000	350.0101	353.5	5	2.512402	2.49975	2.512402	0.012652	10	0.126519	100	1.26519	1000	12.6519	
	Nominal	3000000	350.0151	350	5	2.499946	2.49975	2.499946	0.000196	10	0.001961	100	0.019614	1000	0.196136	
30 bar	**Lower**	3000000	350.0151	346.5	5	2.487383	2.49975	2.487383	-0.01237	10	-0.12367	100	-1.23667	1000	-12.3667	
	Higher	3000000	350.0151	353.5	5	2.512384	2.49975	2.512384	0.012634	10	0.126339	100	1.263395	1000	12.63395	
	Nominal	4000000	350.0201	350	5	2.499928	2.49975	2.499928	0.000178	10	0.001782	100	0.017818	1000	0.178182	
40 bar	**Lower**	4000000	350.0201	346.5	5	2.487365	2.49975	2.487365	-0.01238	10	-0.12385	100	-1.23846	1000	-12.3846	
	Higher	4000000	350.0201	353.5	5	2.512366	2.49975	2.512366	0.012616	10	0.12616	100	1.261599	1000	12.61599	
	Nominal	5000000	350.0251	350	5	2.49991	2.49975	2.49991	0.00016	10	0.001602	100	0.016023	1000	0.160228	
50 bar	**Lower**	5000000	350.0251	346.5	5	2.487347	2.49975	2.487347	-0.0124	10	-0.12403	100	-1.24026	1000	-12.4026	
	Higher	5000000	350.0251	353.5	5	2.512348	2.49975	2.512348	0.012598	10	0.12598	100	1.259804	1000	12.59804	
	Nominal	6000000	350.0302	350	5	2.499892	2.49975	2.499892	0.000142	10	0.001423	100	0.014227	1000	0.142275	
60 bar	**Lower**	6000000	350.0302	346.5	5	2.487329	2.49975	2.487329	-0.01242	10	-0.12421	100	-1.24205	1000	-12.4205	
	Higher	6000000	350.0302	353.5	5	2.51233	2.49975	2.51233	0.01258	10	0.125801	100	1.258009	1000	12.58009	
	Nominal	7000000	350.0352	350	5	2.499874	2.49975	2.499874	0.000124	10	0.001243	100	0.012432	1000	0.124321	
70 bar	**Lower**	7000000	350.0352	346.5	5	2.487312	2.49975	2.487312	-0.01244	10	-0.12438	100	-1.24385	1000	-12.4385	
	Higher	7000000	350.0352	353.5	5	2.512312	2.49975	2.512312	0.012562	10	0.125621	100	1.256214	1000	12.56214	
	Nominal	8000000	350.0402	350	5	2.499856	2.49975	2.499856	0.000106	10	0.001064	100	0.010637	1000	0.106368	
80 bar	**Lower**	8000000	350.0402	346.5	5	2.487294	2.49975	2.487294	-0.01246	10	-0.12456	100	-1.24564	1000	-12.4564	
	Higher	8000000	350.0402	353.5	5	2.512294	2.49975	2.512294	0.012544	10	0.125442	100	1.254418	1000	12.54418	
	Nominal	9000000	350.0452	350	5	2.499838	2.49975	2.499838	8.84E-05	10	0.000884	100	0.008842	1000	0.088415	
90 bar	**Lower**	9000000	350.0452	346.5	5	2.487276	2.49975	2.487276	-0.01247	10	-0.12474	100	-1.24744	1000	-12.4744	
	Higher	9000000	350.0452	353.5	5	2.512276	2.49975	2.512276	0.012526	10	0.125262	100	1.252623	1000	12.52623	
	Nominal	10000000	350.0503	350	5	2.49982	2.49975	2.49982	7.05E-05	10	0.000705	100	0.007046	1000	0.070463	
100 bar	**Lower**	10000000	350.0503	346.5	5	2.487258	2.49975	2.487258	-0.01249	10	-0.12492	100	-1.24923	1000	-12.4923	
	Higher	10000000	350.0503	353.5	5	2.512258	2.49975	2.512258	0.012508	10	0.125083	100	1.250828	1000	12.50828	

Table 3.14 summarised the data in Table 3.13.

Table 3.14: Summary of IA Vout for different Pressure Ranges

Levels --> in Volts	Nom (mV)	Low (mV)	High(mV)
ΔIA Vout for 0bar to 10bar	17.9549	17.9544	17.9544
ΔIA Vout for 10bar to 20bar	17.9546	17.9542	17.9542
ΔIA Vout for 20bar to 30bar	17.9544	17.9539	17.9539
ΔIA Vout for 30bar to 40bar	17.9541	17.9536	17.9537
ΔIA Vout for 40bar to 50bar	17.9538	17.9534	17.9534
ΔIA Vout for 50bar to 60bar	17.9536	17.9531	17.9531
ΔIA Vout for 60bar to 70bar	17.9533	17.9529	17.9529
ΔIA Vout for 70bar to 80bar	17.9531	17.9526	17.9526
ΔIA Vout for 80bar to 90bar	17.9528	17.9523	17.9524
ΔIA Vout for 90bar to 100bar	17.9526	17.9521	17.9521

The voltage for 10bar pressure ranges were consistently ≈17.95 mV. The ADC of the microcontroller had to recognise one level at 20 mV. The Arduino MEGA was investigated and Table 3.15 analysed its ADC capabilities.

Table 3.15: Arduino Resolution Calculations

ADC and Resolutions	
10	**bit**
1024	**Levels**
5	**Vin**
0.004882813	**Volt per level**
4.8828125	**mV per level**
5	**1 Arduino level (mV)**
18	**1 10bar level (mV)**
3.6	**Arduino level per 10bar**

The ADC had a resolution of 5mV per level which confirmed that 10bar intervals could be read but the accuracy was limited.

The INA125 IA amplified the bridge output. It used a single supply and had a built-in power reference for the Vin supply of the bridge (Burr-Brown, 2009). The gain was controlled with a potentiometer. The schematic layout of the INA125 was shown in Figure 3.5.

Image removed due to 3rd party copyright.

Figure 3.5: INA125 Interior Schematic (ibid)

The gain was calculated with Equation 3.6:

$$G = 4 + \frac{60 * 10^3}{R_G} \qquad \text{(Eq. 3.6)}$$

R_G was the potentiometer. For a gain of 1000 the resistance was 60.24Ω. Practically, the potentiometer was adjusted until the voltage value at 0bar represented the correct voltage from the equation. The before and after voltages were also checked. The data collected from the calculations created useful graphs. Figure 3.6 showed the strain and resistance changes at set pressures.

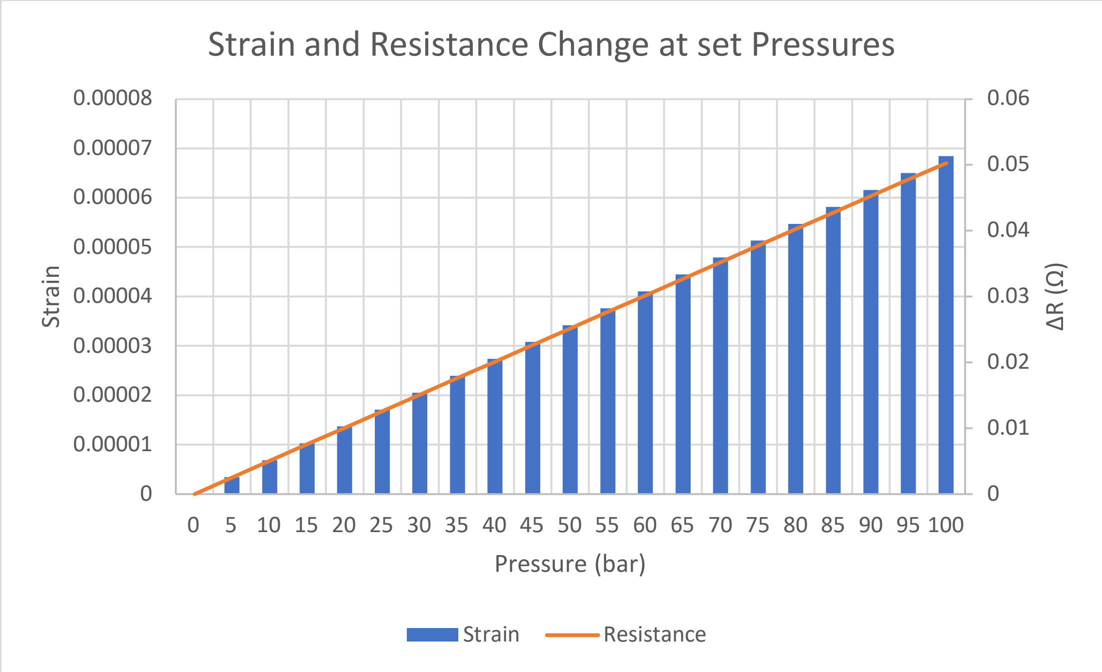


Figure 3.6: Strain and Resistance Change Graph

There was a clear linear correlation between an increasing pressure and an increasing strain. Table 3.16 contained strain values, pressure values and the expected theoretical voltage output from the IA with a gain of 1000.

Table 3.16: Collated Values of Strain, Pressure and Voltage Output

Strain	Pressure (bar)	IA Vout (V)
0	0	0.250
6.84E-06	10	0.232
1.37E-05	20	0.214
2.05E-05	30	0.196
2.74E-05	40	0.178
3.42E-05	50	0.160
4.1E-05	60	0.142
4.79E-05	70	0.124
5.47E-05	80	0.106
6.16E-05	90	0.088
6.84E-05	100	0.070

The data was formatted as a graph in Figure 3.7.

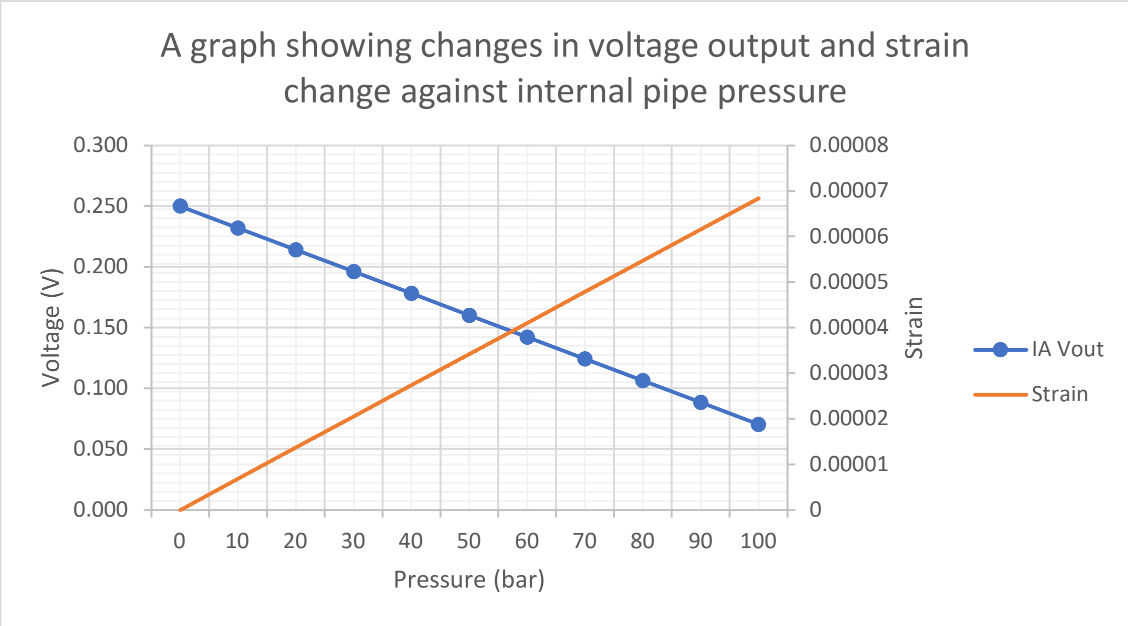


Figure 3.7: Voltage Output, Strain Change vs Internal Pipe Pressure Graph

Both relations were linear but with different gradients. The microcontroller needed to use the voltage level to calculate the pressure, so the graph was distilled further per Figure 3.8 below.

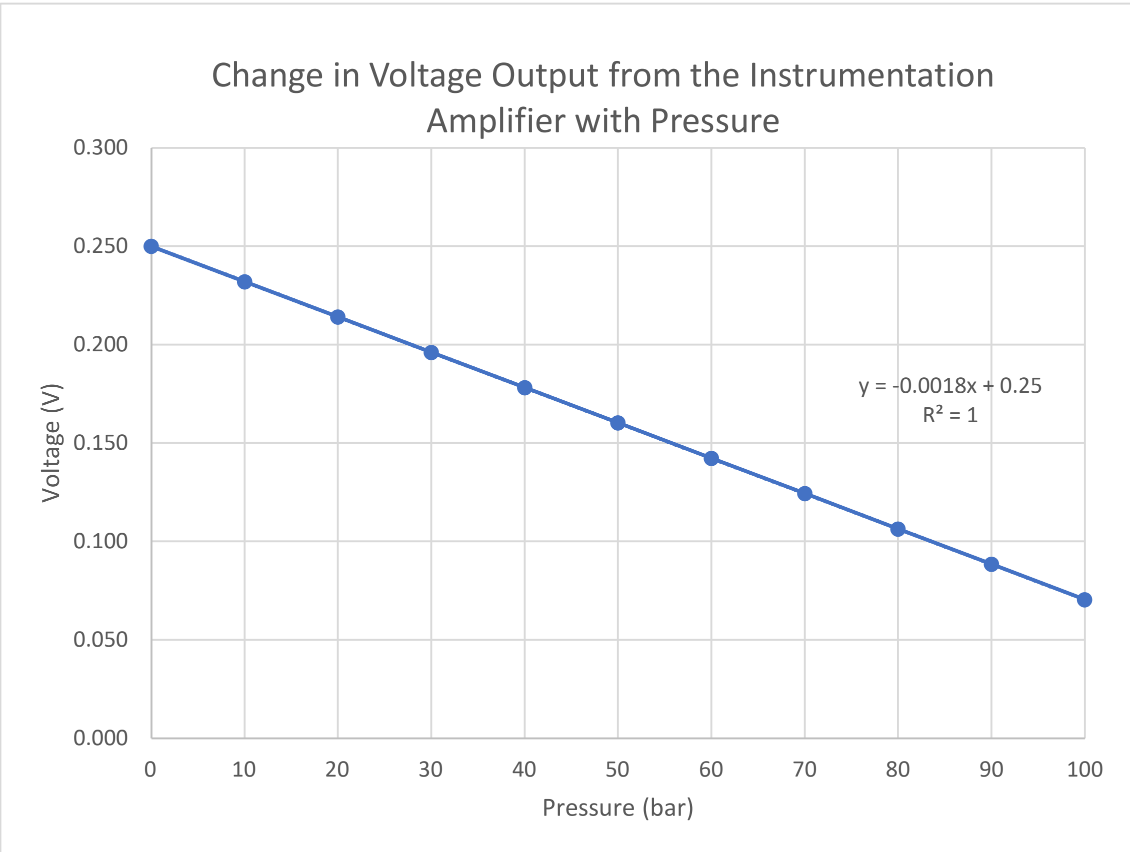


Figure 3.8: Change in Voltage Output with Pressure Graph

The equation for the relationship between the voltage and the pressure was derived by the software, and R^2 was 1. Therefore, the relationship between the voltage and the internal pressure was found by Equation 3.7:

$$V = (-0.0018 * P) + 0.25 \qquad (\mathrm{Eq.}\ 3.7)$$

Equation 3.7 was adapted so pressure was the subject and the voltage variable were changed to the input level multiplied by 0.004882813 – the same as the C&B sensor – per Equation 3.8:

$$P = \frac{(V_{lvl} * 0.004882813) - 0.25}{-0.0018} \qquad (\mathrm{Eq.}\ 3.8)$$

This equation was used in the microcontroller, the program code was added to Appendix A6. To begin, all of the inputs and outputs were defined. The strain gauge circuit used the A0 analog input and the lifebit functions were programmed in the same manner as the C&B program. There were also the calibration input and calibration completion signal too. The additions to this program were the 'pumpson' input and the 'bracketone' to 'bracketten' outputs. The former dictated when calibration began and when measurements occurred. The outputs from 'bracketone' to 'bracketten' communicated to the PLC which pressure range the sensor measured. A trigger for large exceeded limits was included to minimise interferences.

All of the pins were declared as inputs or outputs and then the main loop started; the program would not run unless it received a lifebit from the PLC. The calibration was only completable if the pumps were off; enabling a 0bar calibration. When the pumps were on and the 'calibcompinternal' bit was high then 3 sets of 10 readings of pressure were taken and then averaged with all 3 required to show pressures within the same bracket. A series of 'if' statements checked the average pressure and activated the relevant output bits for the correct bracket. If these pressure ranges were positively exceeded then only the Limit Exceed was set. An error signal was sent if the pressure was less than 0bar. The programming methodology was kept the same as the C&B program for ease of fault finding and it created a platform for future expansion.

The block diagram was showed in Figure 3.9 below.

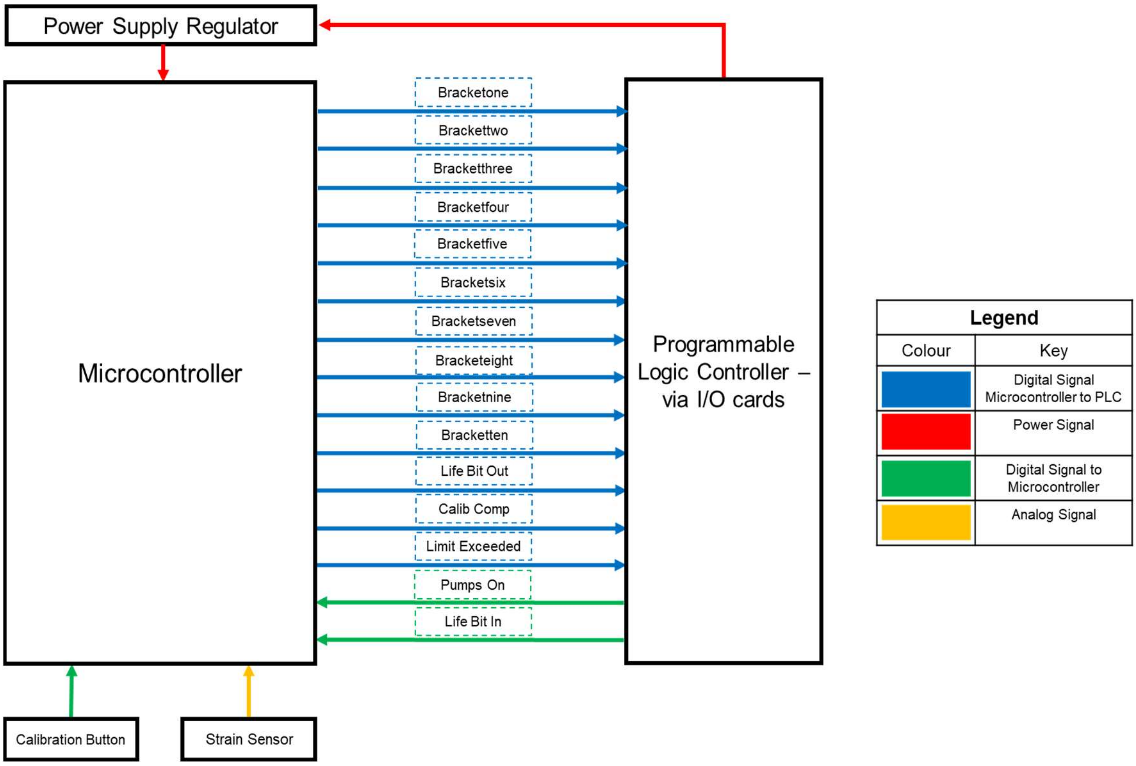


Figure 3.9: NIPP Sensor Block Diagram

More interfacing connections and regulators were required due to the circuit setup. The wire lengths between the strain gauges and microcontroller were kept to a minimum with very low resistances recorded to prevent errors, equation offsets were a backup tool for error correction.

The INA125 IC provided a 5V reference voltage to the bridge; to do this it needed its own 12V regulator. As gain increased so did the CMRR (*ibid*), with a gain of 1000 there was good rejection of interference. The circuit schematic for interfacing and calibration was added to Appendix A7 and the measuring circuit schematic was placed in Appendix A8. The first part of the circuit contained a polarity protected 12V regulator for the IA. The strain gauges and resistors were setup into a bridge configuration. The bridge fed the V+ and V- connections on the IA. A 1kΩ potentiometer was used for gain control. The output signal came from the connection between 'Sense' and Vo' as described by the manufacturer. Decoupling capacitors were used throughout the circuit to reduce interferences and reduce ripple on the voltage output.

3.6 Power, Interfacing Circuits and Wireless Transmission

3.6.1 Power to the Sensors

All circuits received power from the PLC 24V and 0V supply. Switching regulators and an isolated DC-DC converter were used with capacitors and provided a stable DC supply. A schematic of the regulation circuits was drawn in Figure 3.10.

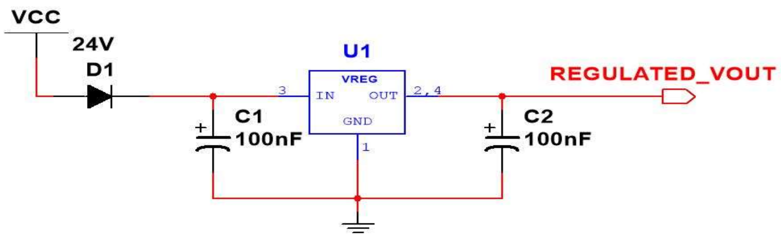


Figure 3.10: General Voltage Regulator Setup

The RECOM switching regulator series supplied 5V signals at 0.5A (Recom, 2014) and for the Arduino power they provided 9V up to 1A. The Raspberry Pi was fed from a 5V 4A isolated DC-DC converter. The 9V regulator circuit schematic was added to Appendix A9.

3.6.2 24V Digital PLC Signal to 5V/3.3V Digital Microcontroller Signal

A potential divider stepped down the 24V signal to a 5V signal, as shown in Figure 3.11.

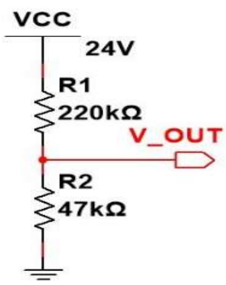


Figure 3.11: Potential Divider for 24V to 5V

The values for the divider were derived from Equation 3.9, (Horowitz and Hill, 2015).

$$V_{OUT} = V_{IN} * \frac{R_2}{R_1+R_2} = \frac{V_{IN}*R_2}{R_1+R_2} \text{ (Eq. 3.9)}$$

The voltage output equation was derived with R1 as 220kΩ and R2 as 47kΩ, per Equation 3.10.

$$V_{OUT} = 24 * \frac{47k}{220k + 47k} = 4.22V \qquad \text{(Eq. 3.10)}$$

The Raspberry Pi GPIO was 3.3V tolerant, not 5V. For consistency, the same resistors were used as for the 5V signals but an additional 100kΩ resistor was placed in parallel with the 47kΩ resistor: the equivalent parallel resistance was 31.973kΩ. Using the equations above, the resultant voltage was 3.045V. The divider schematic was added to Figure 3.12.

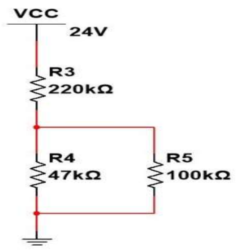


Figure 3.12: Potential Divider for 24V to 3.3V

3.6.3 5V/3.3V Digital Microcontroller Signal to 24V Digital PLC Signal

An optocoupler switched a 24V supply through a current limiting resistor to a PLC input. Figure 3.13 showed this setup:

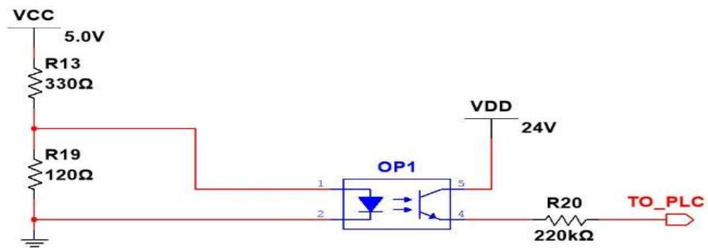


Figure 3.13: Switching Optocoupler System

The chosen optocoupler was an FOD817 Series 4-Pin Phototransistor Optocoupler. The input voltage for this optocoupler was limited to 1.4V (Fairchild Semiconductor, 2011). The divider stepped 5V down to 1.4V for the emitter; the calculations showed a theoretical voltage of 1.33V. For the VA circuit the supply was 3.3V instead of 5V and so 220Ω resistors replaced the 120Ω resistors and this provided 1.32V to the optocoupler. The voltages needed to be close to 1.4V to enable full switching capacity.

3.6.4 Wireless Transmission

The nRF24L01 module (shown in Figure 3.14) was the chosen wireless transmission method.

Image removed due to 3rd party copyright.

Figure 3.14: nRF24L01 Module Diagram (Components101, 2018)

The transceiver used SPI at 2MBPs to communicate with the microcontroller (Nordic Semiconductor, 2008). The supply required was 3.3V and the inputs were 5V tolerant. A prefabricated breakout board (Figure 3.15) was used because it contained a 5V to 3.3V regulator for powering the transceiver.

Image removed due to 3rd party copyright.

Figure 3.15: nRF24L01 Breakout Board (Cytron, no date)

The pins on this were: CE, CSN, SCK, MOSI, MISO and IRQ. CE was the Chip Enable pin, CSN was the Chip Select pin, SCK was the clock pin, MOSI was the Slave Data Input pin, MISO was the Slave Data Output pin and IRQ was an Interrupt pin. The IRQ pin was not used. The SPI system operated as follows: the SCK signal was the master timing signal sent from the master device to all slaves – it was used for synchronisation. When the master device requested data from a slave device it set CSN low and then took data from MISO while placing data on MOSI (Leens, 2009).

Each sensor was designed to send data as a string or characters. All transceivers were connected to Arduino MEGA 2560 microcontrollers. Both the C&B and NIPP sensors already used the Arduino so the transceiver was added to the same controller. The VA sensor and CGD used a Raspberry Pi so an additional Arduino MEGA 2560 was used with each Raspberry Pi and communication was established via USB. This configuration ensured that Arduinos would only wirelessly communicate with other Arduinos which reduced programming complexity. The block diagram seen in Figure 3.16 showed the setup of the wireless system.

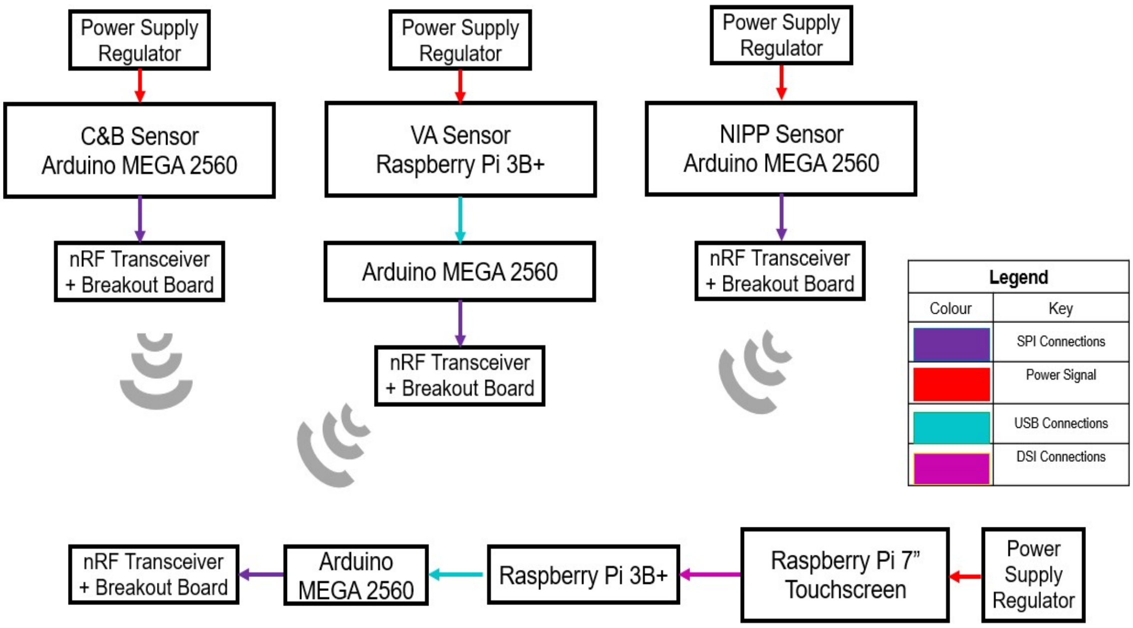


Figure 3.16: nRF Wireless System Block Diagram

Each sensor system was assigned an address to send and receive data – in nRF terms this was known as the 'pipe'. Each sensor sent data down its pipe as often as required; it was the task of the receiver connected to the CGD to decode the information. The frequency of transmission was based on 2.4GHz. The sensors sent unique text so that the data origin was known which reduced misreporting. The devised transmission messages were summarised in Table 3.17.

Table 3.17: Wireless Network Process Summary

Sensor	**C&B**	**VA**	**NIPP**
Main Controller	Arduino	Raspberry Pi	Arduino
Message: OK?	1OK	1 (2OK)	N/A
Message: Wearing?	1WE	2 (2WE)	N/A
Message: NOK?	1NO	3 (2NO)	N/A
Message: Limit Exceeded?	1LE	4 (2LE)	3LE
Message: Error?	1ER	5 (2ER)	3ER
Message: Other	N/A	N/A	301; 302; 303; 304; 305; 306; 307; 308; 309; 310
Conversion needed?	NO	YES	NO

The VA sensor required conversion because it sent the data in ASCII character format – which was translated by the Arduino into a decimal equivalent which ensured consistency with the other sensors. Table 3.18 summarised the decoding required.

Table 3.18: ASCII to DEC Conversion (WatElectronics, 2019)

ASCII	**Decimal**
1	49
2	50
3	51
4	52
5	53

These decimal numbers were then used in conditional loops to choose the correct message. Each message contained 3 characters. The first character contained the sensor number: 1 for C&B, 2 for VA and 3 for NIPP. The next two characters were OK, WE, NO, LE or ER for the C&B and VA sensors. The NIPP sensor used pressure bracket numbers for the last two characters.

The C&B wireless transmission program was added to Appendix A10, it was written in the Arduino IDE and 3 libraries related to the nRF device were used from the tmrh20 repository (Tmrh20, 2020) which provided helpful guidance and information. The first stage setup the CE and CSN pins, then an address was assigned to the microcontroller – “00001”. During the setup loop radio communications initialised, the address pipe was opened, the power amplifier level was set and the device was told to stop listening; allowing it to transmit. The main loop sent the data. A variable was assigned the value “1NO” to represent a condition and then the data in this array was transmitted over RF. The template for the NIPP program, Appendix A11, was nearly identical excluding the message being sent – “306” representing a pressure of 50-60bar - and the sensor address.

The VA sensor wireless data transmission programs started with the Raspberry Pi program written in Python in Appendix A12. Two libraries were imported: serial for the serial communication and time for time functions. The variable ‘ser’ was assigned to the serial data coming from the USB port identified as ‘ttyACM0’ and the data rate was set to 9600. There was a 5 second pause and the serial bus was flushed. A while loop was setup to send a character: in this example it was ‘1’ which corresponded to “2OK”. The second program – the Arduino part - was added to Appendix A13. The initial setup was consistent with pins declared, libraries declared and a unique address provided. An integer variable ‘r’ was initialised as ‘0’. The serial bus was initialised in the setup loop because the device needed to communicate via USB to the Pi. In the main loop of the program, an if statement checked if there was any serial connection available – if there was then the data was stored in the integer variable, r. Following this a series of if statements decoded the number sent by the Pi. The correct message was then assigned to an array and wirelessly transmitted.

The next program was for the Arduino connected to the Raspberry Pi of the CGD. The three sensors transmitted data on their own accord and the receiver was designed to scan through each pipe in turn, take the data from the pipe, store it locally and then combine all of the data into one variable. This program was included as Appendix A14. The program began with declaration of libraries and pins. This program was designed with future expansion in mind so a variable called ‘numSlaves’ was declared and set to ‘3’ – it could have increased in the future. The addresses were then stated and a buffer was setup to hold the data transmitted. In the setup loop the serial setup was initialised alongside all basic radio configurations. The radio was then told to ‘startListening’. In the main loop the first ‘for’ loop cycled through the reading pipes, based on the iterative figure ‘n. Looking at the first reading pipe, it then went through an ‘if’ statement checking if radio data was available; if it was then it read the data and stored it in the relevant recv_buff that was linked to ‘n’. The data in the three separate recv_buff variables was combined into ‘recv_tot string’ and sent to the Raspberry Pi.

The final program was the Raspberry Pi CGD program – in Appendix A15 - which created a GUI. The GUI was generated with the appJar module for Python (appJar, 2018). The GUI was displayed on a 7” Raspberry Pi Touchscreen; the entire device was powered by a portable powerpack with 20000mAH capacity and maximum current output per port of 3A. Firstly, the appJar module was imported and labelled as ‘gui’. A window (the interface) was created, the serial library was imported and the variable ‘ser’ was assigned to the USB port in use. The data was read and stored in the read_serial variable where the string was then decoded. This final message was stored in the string variable ‘rec_buff’ and printed to the program’s shell monitor so that it was visible even without a GUI. Three further buffers ‘chkbuff1/2/3’ held a single character of the string and specifically looked for ‘1’, ‘2’ or ‘3’. Later in the program a series of if statements checked that these buffers – in any order but not repeated – contained these numbers: if it was the case then the loops were broken and the main part of the program was used. The first, second and third sets of ‘threechars’ were collected and stored for future analysis. The long list of conditional ‘if statements’ after the data gathering were used to generate the correct corresponding GUI element. The entire transceiver network, CGD and GUI were designed with features which helped reduce the effect of interferences in the system.

3.6.5 Microcontrollers

The microcontroller and microcomputer requirements were discussed in previous sections. In summary: C&B and NIPP used the Arduino MEGA 2560 while the VA & CGD used the Raspberry Pi 3B+. Both the VA and CGD were connected to 7” touchscreens. The VA and CGD also required an additional Arduino MEGA 2560 for the transceivers.

3.7 Programmable Logic Controller Programming

The program was written in the Siemens SIMATIC S7 environment in ladder logic for use with Siemens PLCs. A Siemens Statement List version was included in Appendix A16; it was converted by the Siemens software automatically.

Network 1 sent a continuous lifebit to the microcontrollers via output ‘Q30.0’. Networks 2-4 were programmed to transfer the state of ‘I30.0’, ‘I40.0’ and ‘I50.0’ into their own local memory bits; identified by ‘M’; as shown in Figure 3.17.

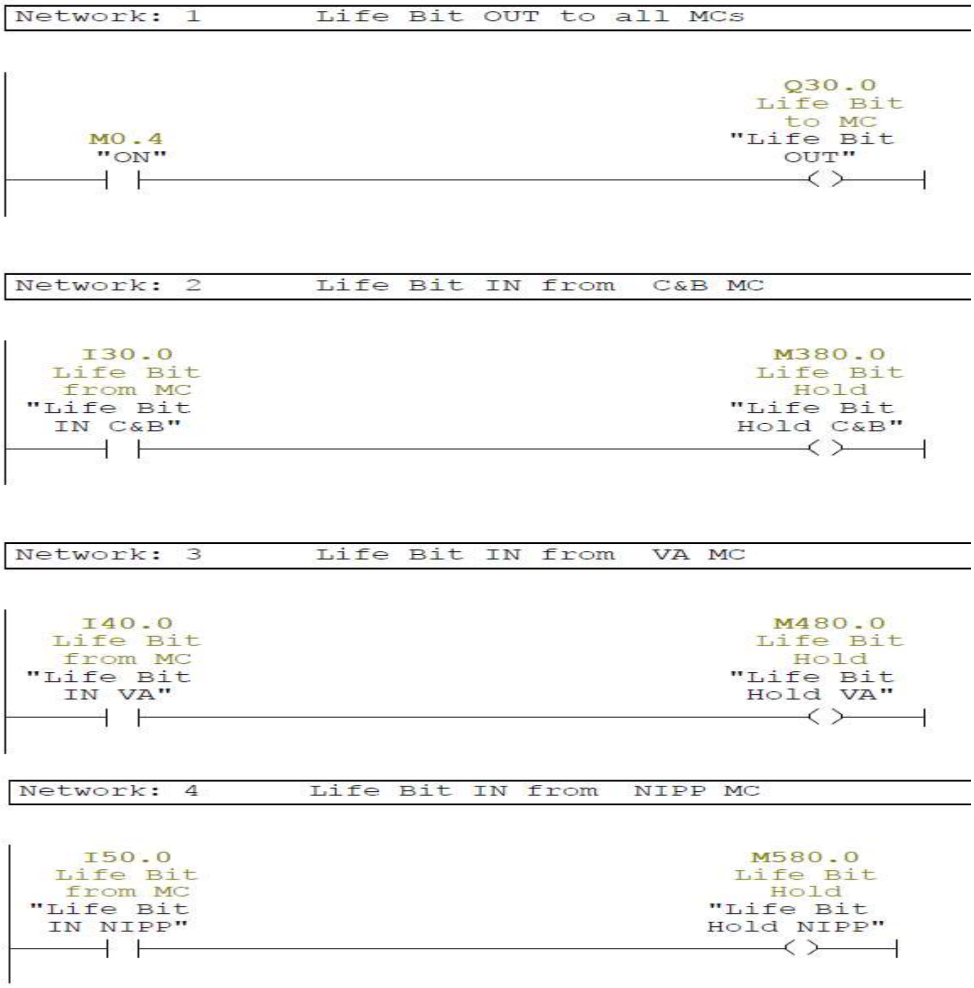


Figure 3.17: PLC Program NW1-4

Networks 5-7 were used to check the lifebits from the sensors were valid. The signal was required to be present for 2s before activating another memory bit; as shown in Figure 3.18.

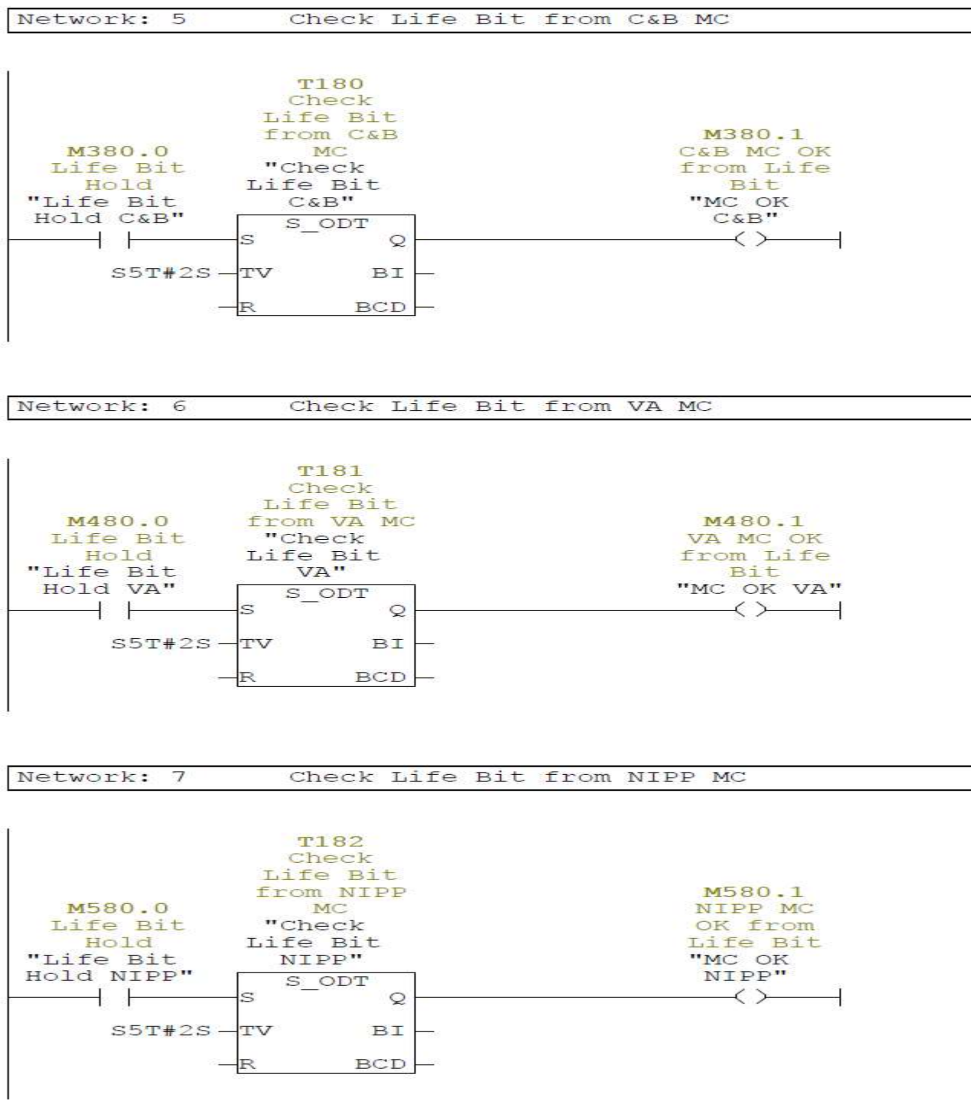


Figure 3.18: PLC Program NW5-7

In network 8 'M390.0' needed to be activated on the PLC by maintenance engineers to allow the program to be used. The subsequent checks looked for 'checked' lifebit signals before 'M390.1' was enabled as showed in Figure 3.19.

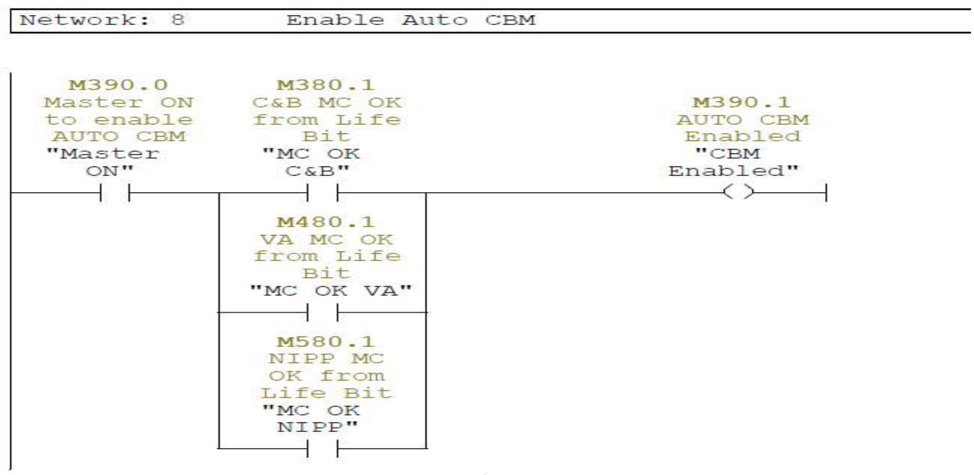


Figure 3.19: PLC Program NW8

Network 9-11 enabled monitoring for each individual sensor as long as 'M390.1' was high and if the sensor was explicitly chosen by the maintenance engineers: per Figure 3.20.

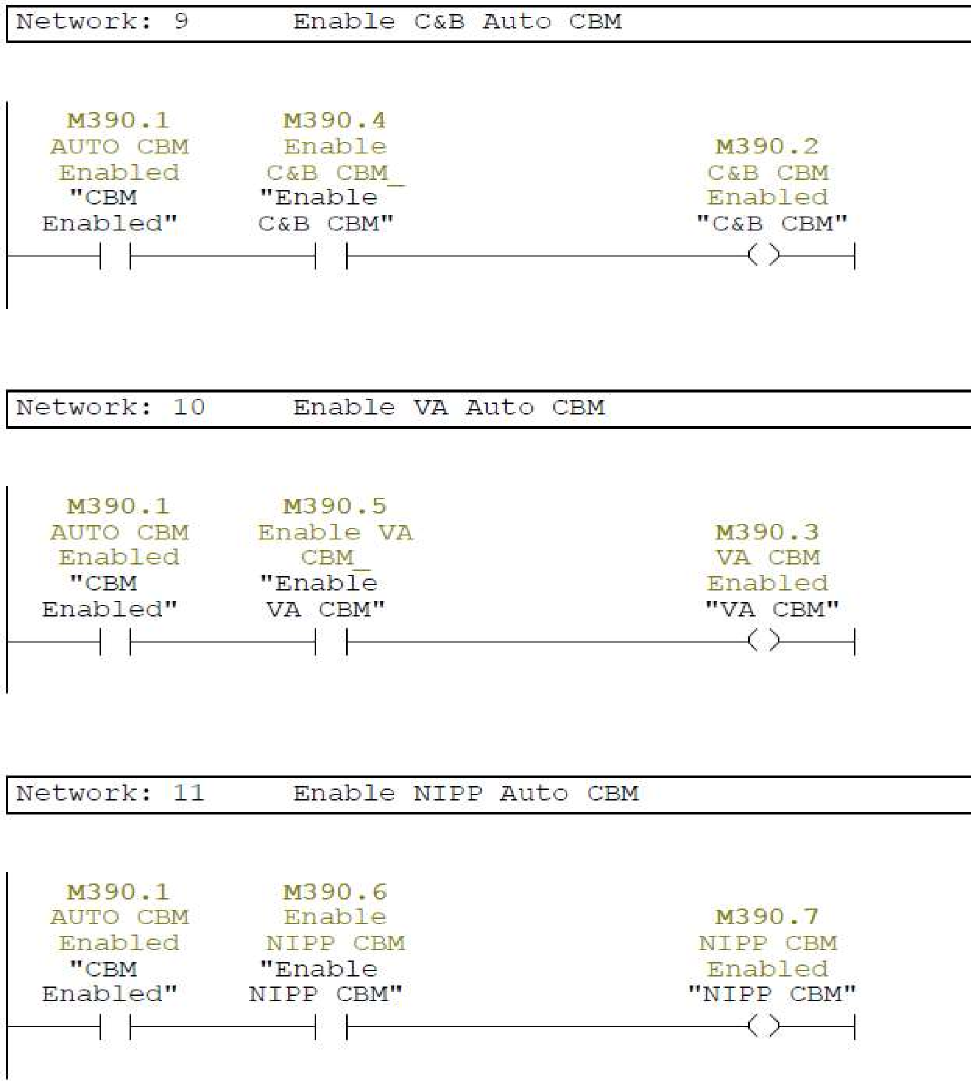


Figure 3.20: PLC Program NW9-11

Network 12-15 focussed on the C&B. They mapped the TDC signal and Press Motive signal from the press to a PLC output and mapped the 'calibration complete' signal from the microcontroller to a memory bit in the PLC, as shown in Figure 3.21.

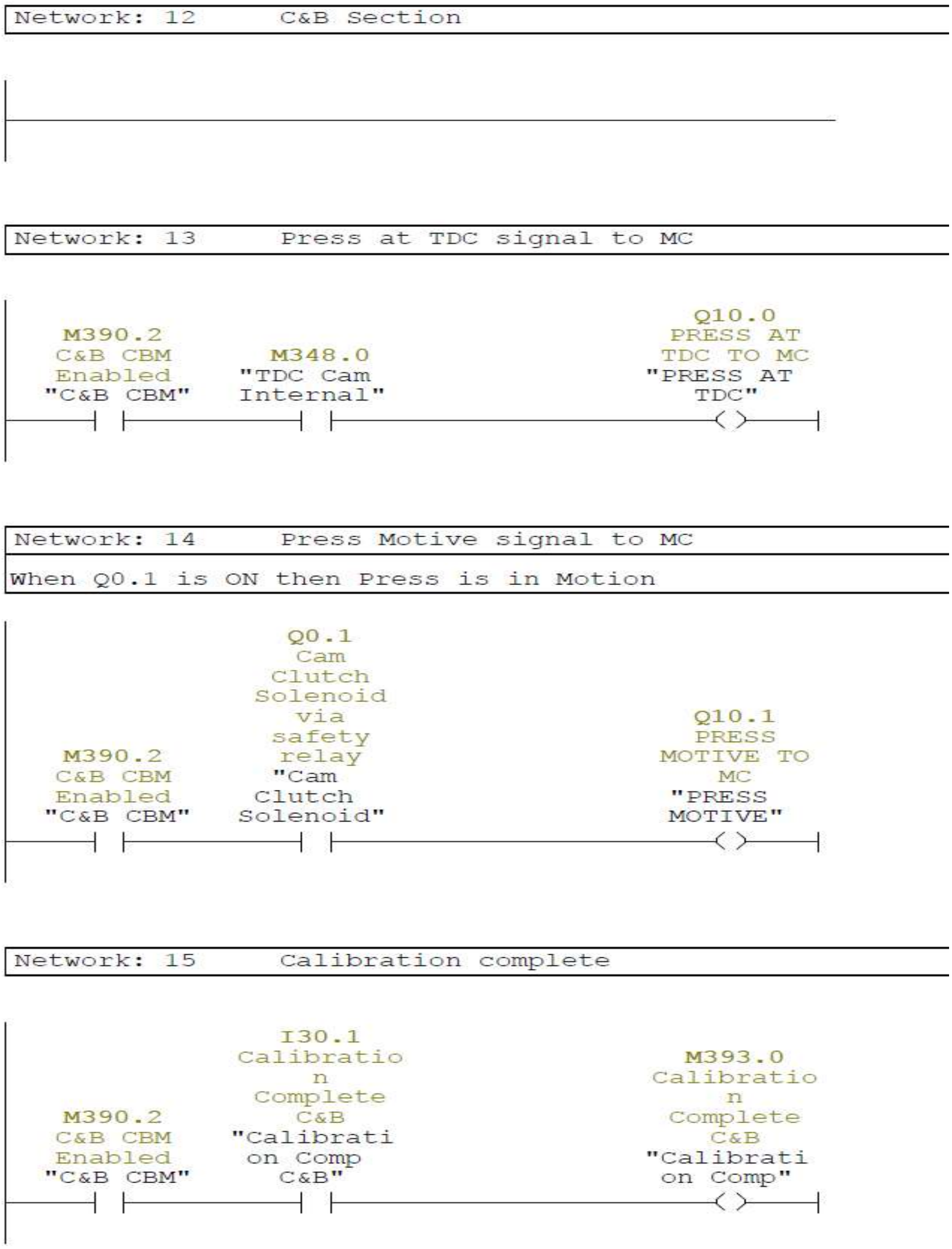


Figure 3.21: PLC Program NW12-15

To map the conditions from the microcontroller to the PLC, network 16 (Figure 3.22) was used. The ‘enable’ was activate and before the inputs were mapped to memory bits the calibration complete signal had to be high. If there was no calibration then the conditions would have been inaccurate.

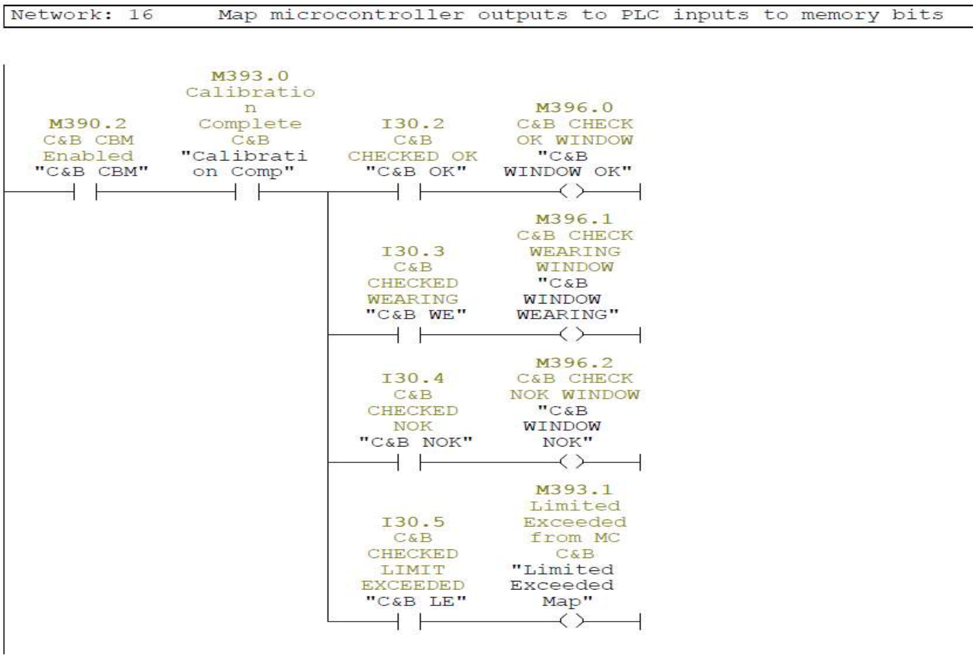


Figure 3.22: PLC Program NW16

Network 17 (Figure 3.23) used the condition memory bits to activate other bits which were solely used as alarms.

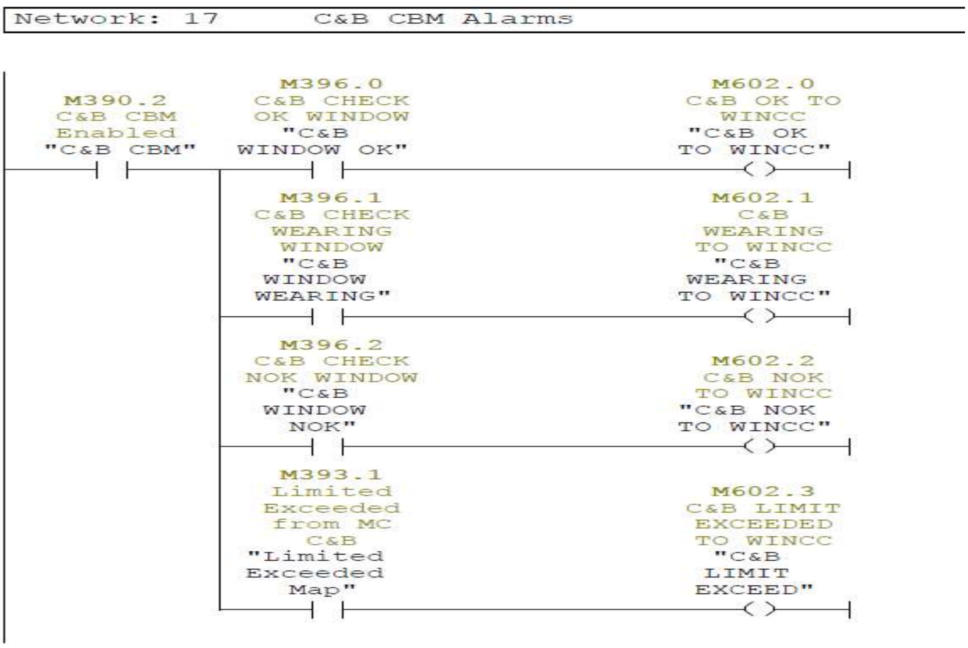


Figure 3.23: PLC Program NW17

Network 18 was the first of two corrective actions programmed if the status was Wearing or NOK. An initial check was performed to ensure that 'C2 Store' was 0 meaning the network was used for the 1st time. Then a comparison checked if the ram brake position was >300°; if it was then it was reduced by 2°. Reducing the ram brake position made the ram brake earlier which reduced the impact of a Wearing or NOK brake pad. Once this corrective action was completed the counter ‘C2’ was increased by 1 and ‘C2 Store’ was updated as explained in Figure 3.24.

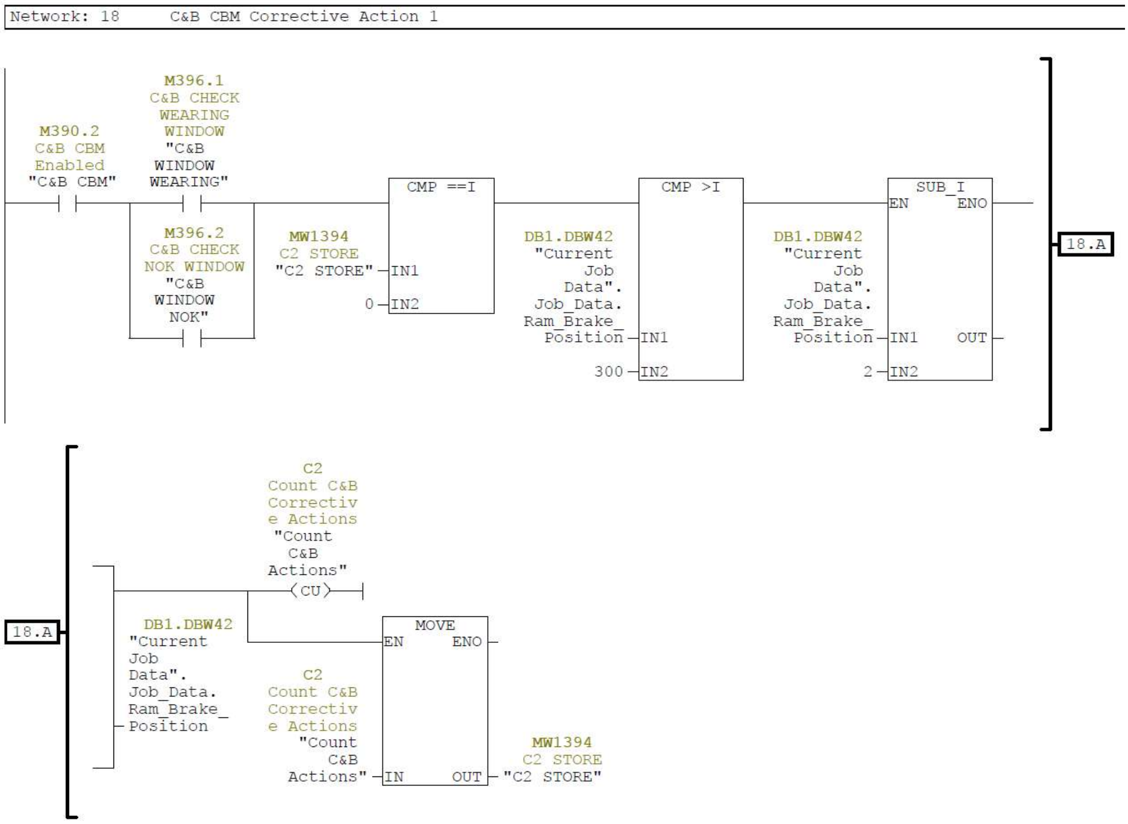


Figure 3.24: PLC Program NW18

Network 19 (Figure 3.25) was the second corrective action, the preconditions were similar to Network 18 but ‘C2 Store’ had to be 1. A timer provided a 15-minute buffer between actions which allowed the previous action to take effect. A check was performed to see if the ‘Ram Lower Window Limit’ was >340°; if so 5°was subtracted. Then the ‘Ram Upper Window Limit’ was checked if it was <20°, adding 5° degrees if it was. The counter was incremented by 1 and ‘C2 store’ updated. The bigger stopping window reduced the frequency of line stoppages – the ultimate stops of 340° and 20° were mechanically safe.

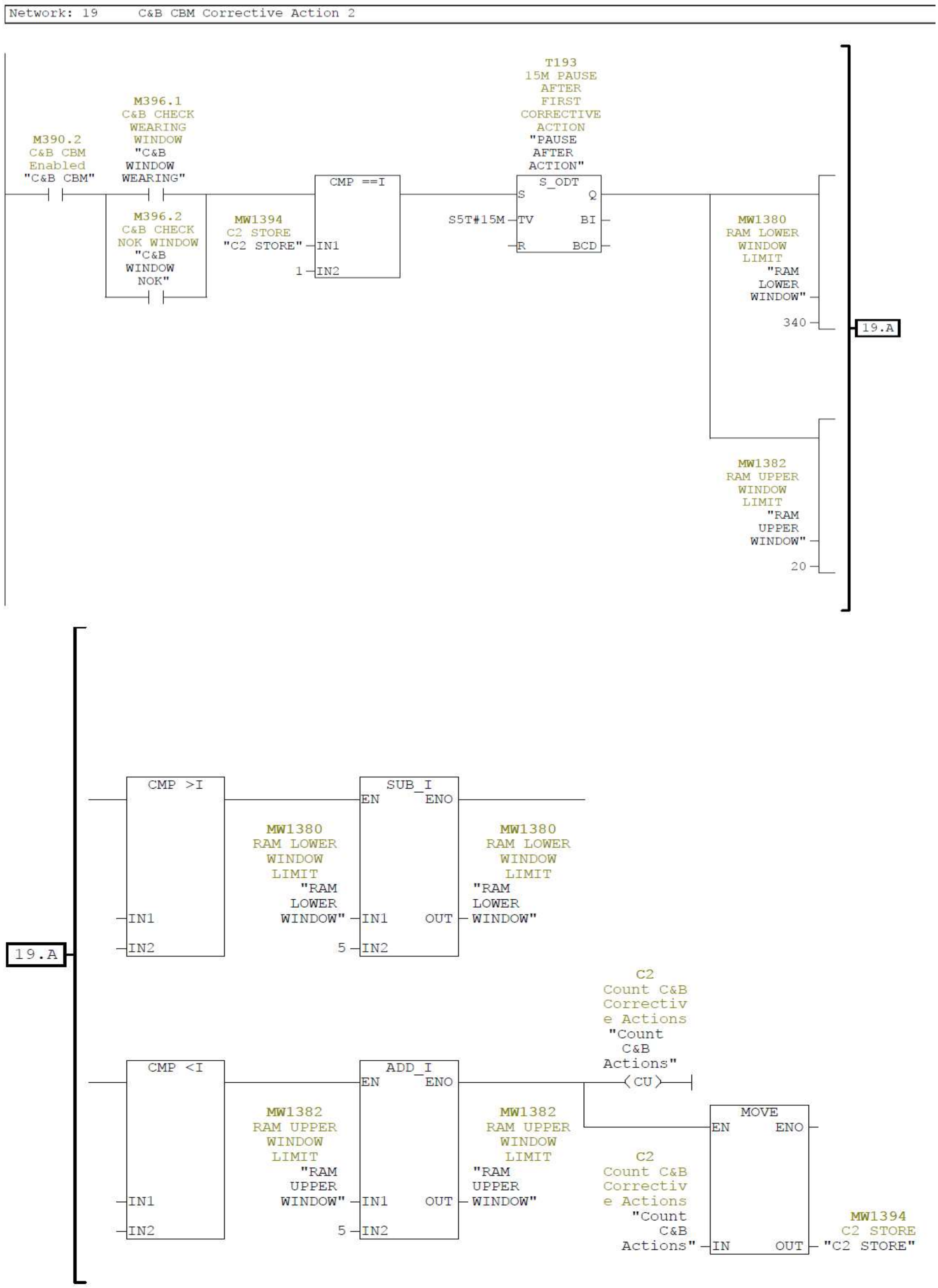


Figure 3.25: PLC Program NW19

Network 20 (Figure 3.26) reset the counter the process could restart when both corrective actions were completed.

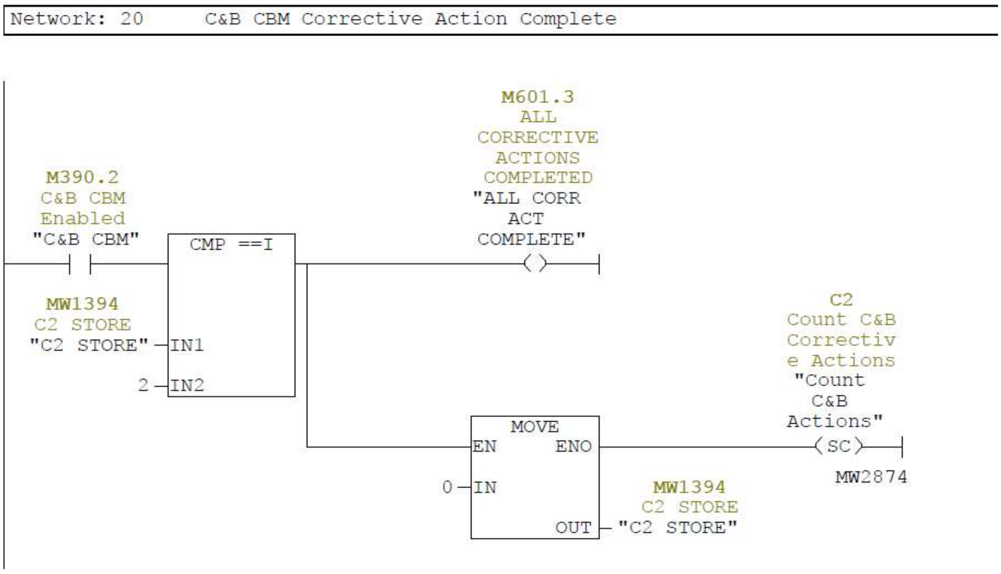
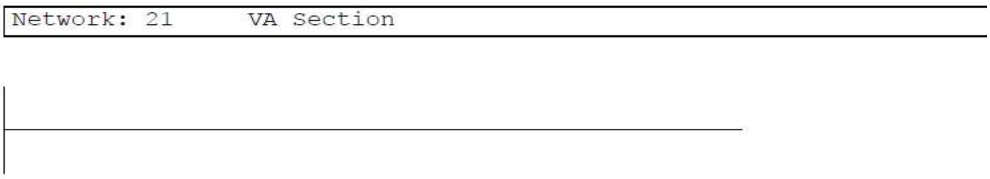


Figure 3.26: PLC Program NW20

Network 21-22 were the beginning of the VA section as showed in Figure 3.27. The programming for precondition requirements was consistent throughout all sensors. Network 22 provided the 'Press at Bottom' signal to the microcontroller. The signal was activated if the ram position was ≥170° and <190°.



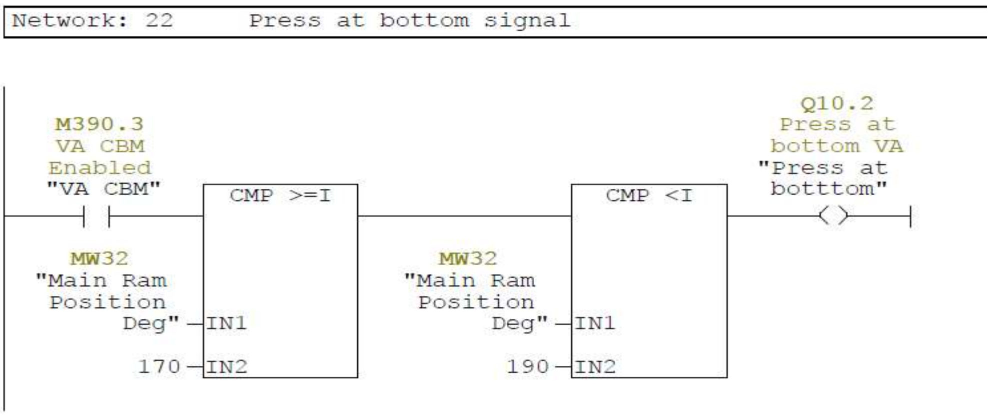


Figure 3.27: PLC Program NW21-22

Network 23 was the VA microcomputer to PLC mapping network (Figure 3.28).

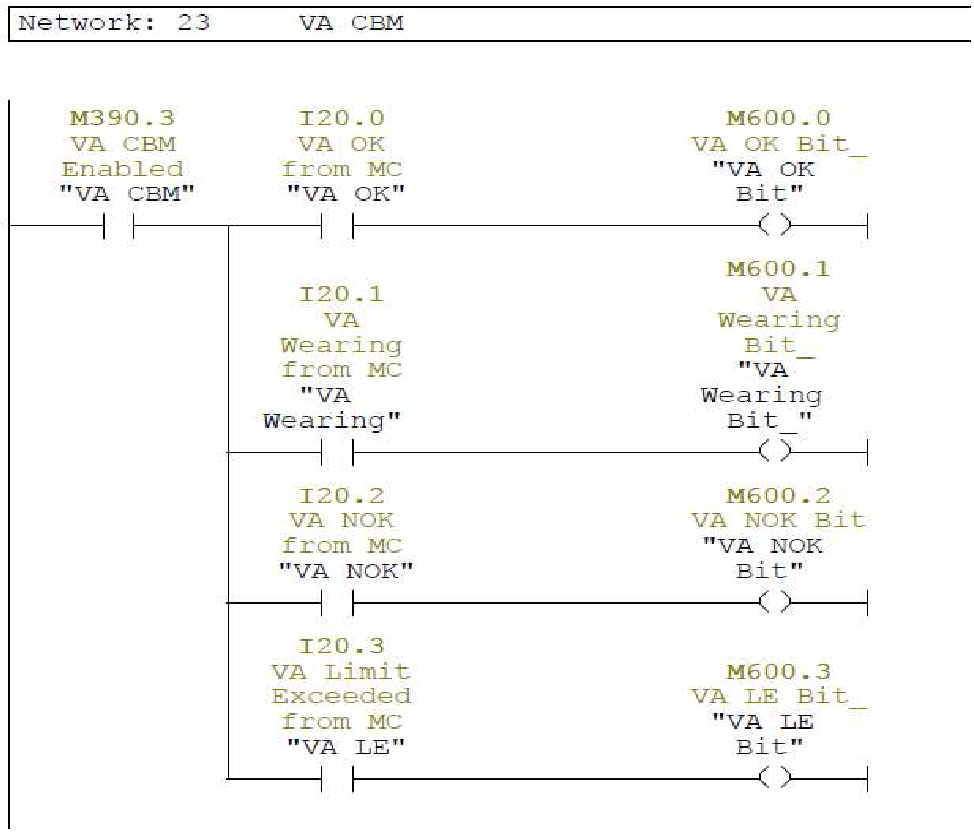


Figure 3.28: PLC Program NW23

Network 24 followed the same pattern as the C&B alarm network by mapping the 4 condition memory bits to 4 specific alarm memory bits as showed by Figure 3.29.

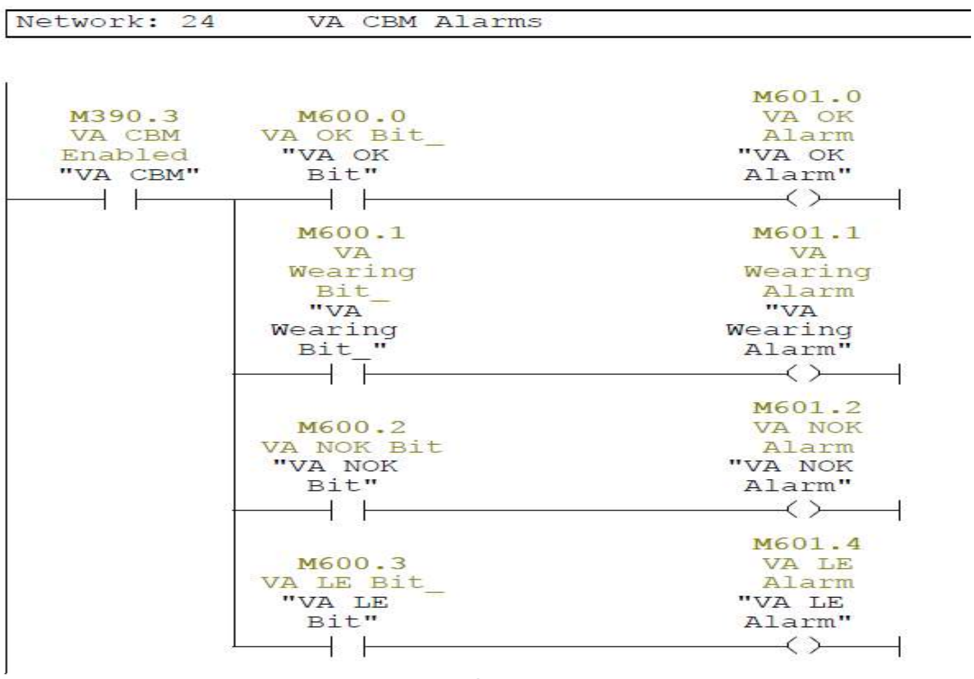


Figure 3.29: PLC Program NW24

The final section was for the NIPP sensor. Network 25-35 showed the mapping of the 'bracket' signals from the microcontroller; one example was shown in Figure 3.30. The networks contained the usual preconditions and input to memory bit mapping; however, a number that represented the upper limit of pressure for the bracket was moved into a memory word for later use.

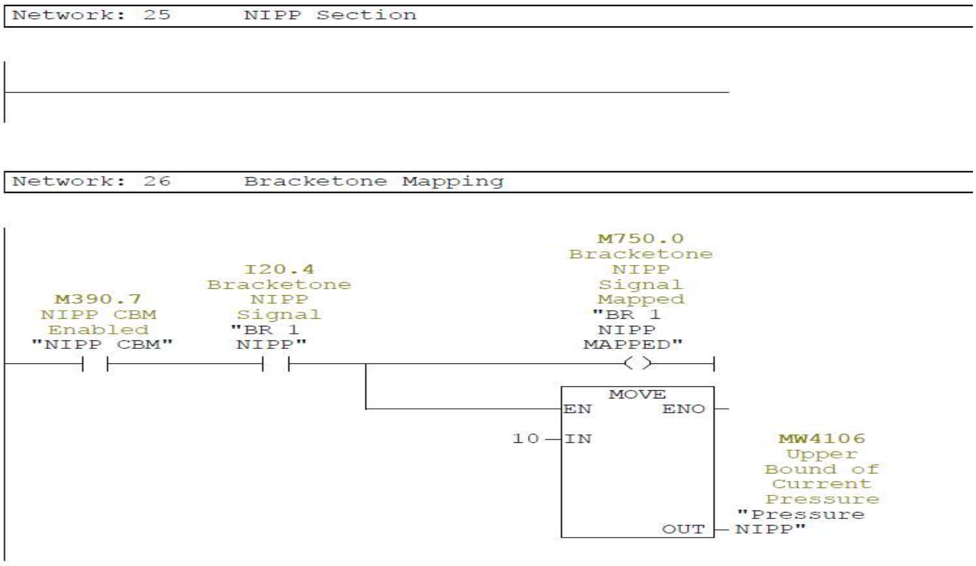


Figure 3.30: PLC Program NW25-26

Network 36-38 were mapping networks (Figure 3.31). The 'Pumps On' signal mapped to an output, 'calibration complete' and 'limit exceeded' signals mapped to memory bits.

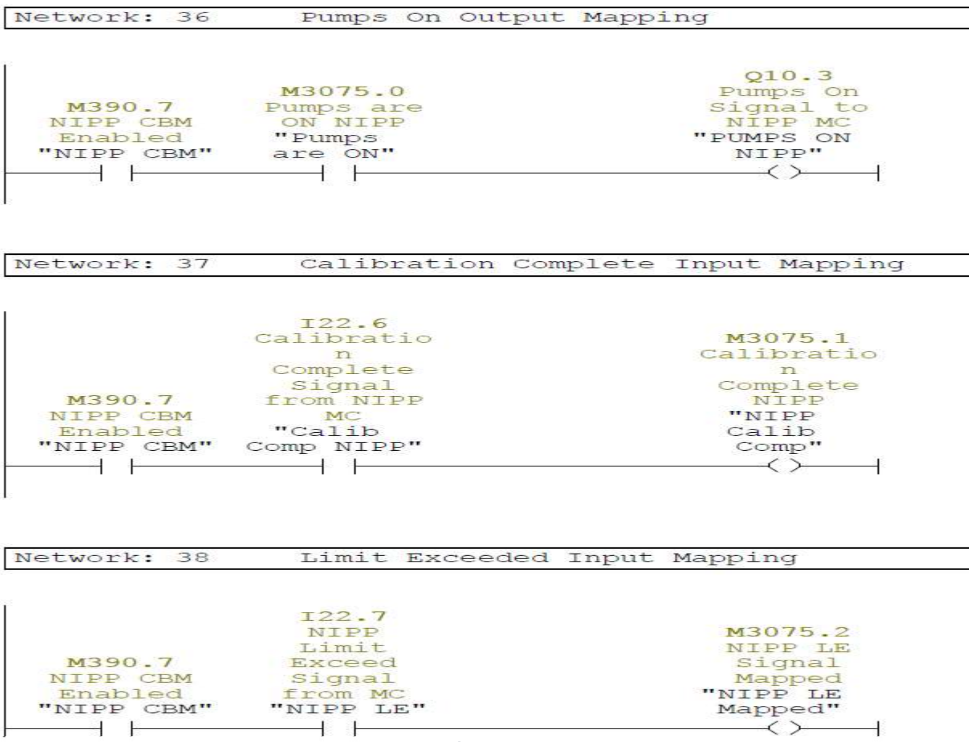


Figure 3.31: PLC Program NW36-38

In Network 39 (Figure 3.32) all of the 'bracket' signal memory bits were mapped to their own alarm bits alongside the 'Limit Exceed' signal.

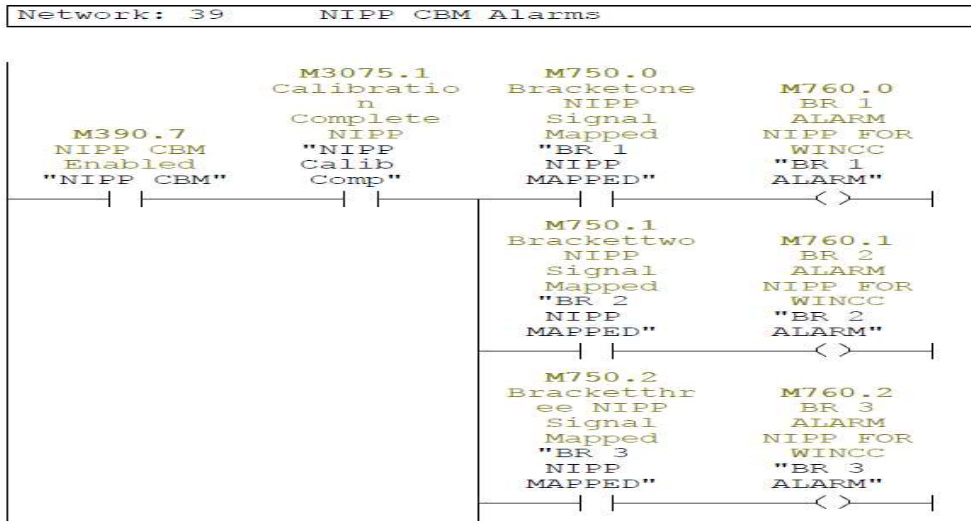


Figure 3.32: PLC Program NW39

Network 40, showed in Figure 3.33, was the first part of the pump shut off sequence. On the first scan the variable 'MW4108' called 'Most recent Pressure Reading this Scan' was 0. The variable 'MW4106' was the previously mentioned 'pressure mapping' variable and as such it had a value other than 0; the ≠ comparison moved the program onward. The result of the division was 0 and therefore the next comparison was false and 'C3' was not incremented. However, when the pressure was tested at 100bar and suddenly dropped to 10bar – giving 10 as a rate of change, then counter 'C3' and 'C3 store' increased to 1.

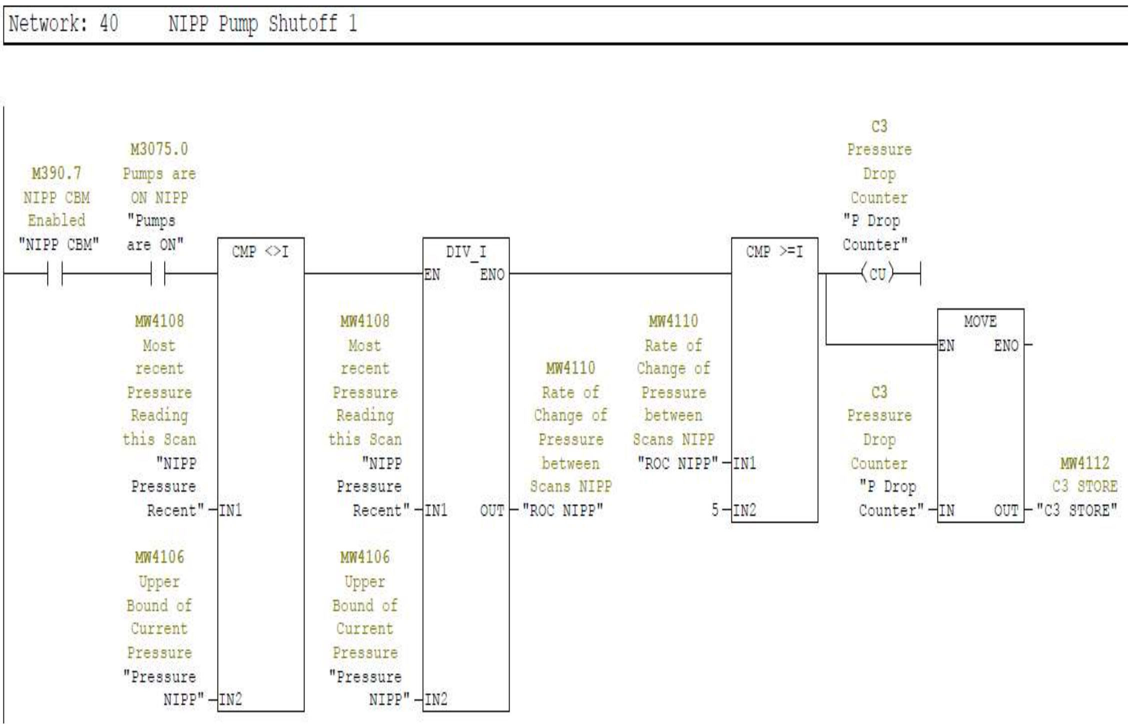


Figure 3.33: PLC Program NW40

Network 41 (Figure 3.34) continued the shutoff. Initially it checked if ‘C3 Store’ equalled 1 and if it did then it checked that the ‘bracketone’ input was active followed by a brief 2 second pause. At this point if there had been a large rate of change and no movement from the lowest pressure bracket for 2 seconds it was classified as a leak. A signal was then sent to cycle stop the press and a flag was raised. An additional 2 second timer provided time for the press to cycle stop properly and for the pumps to shut off. After 2 seconds the ‘C3’ counter reset to 0.

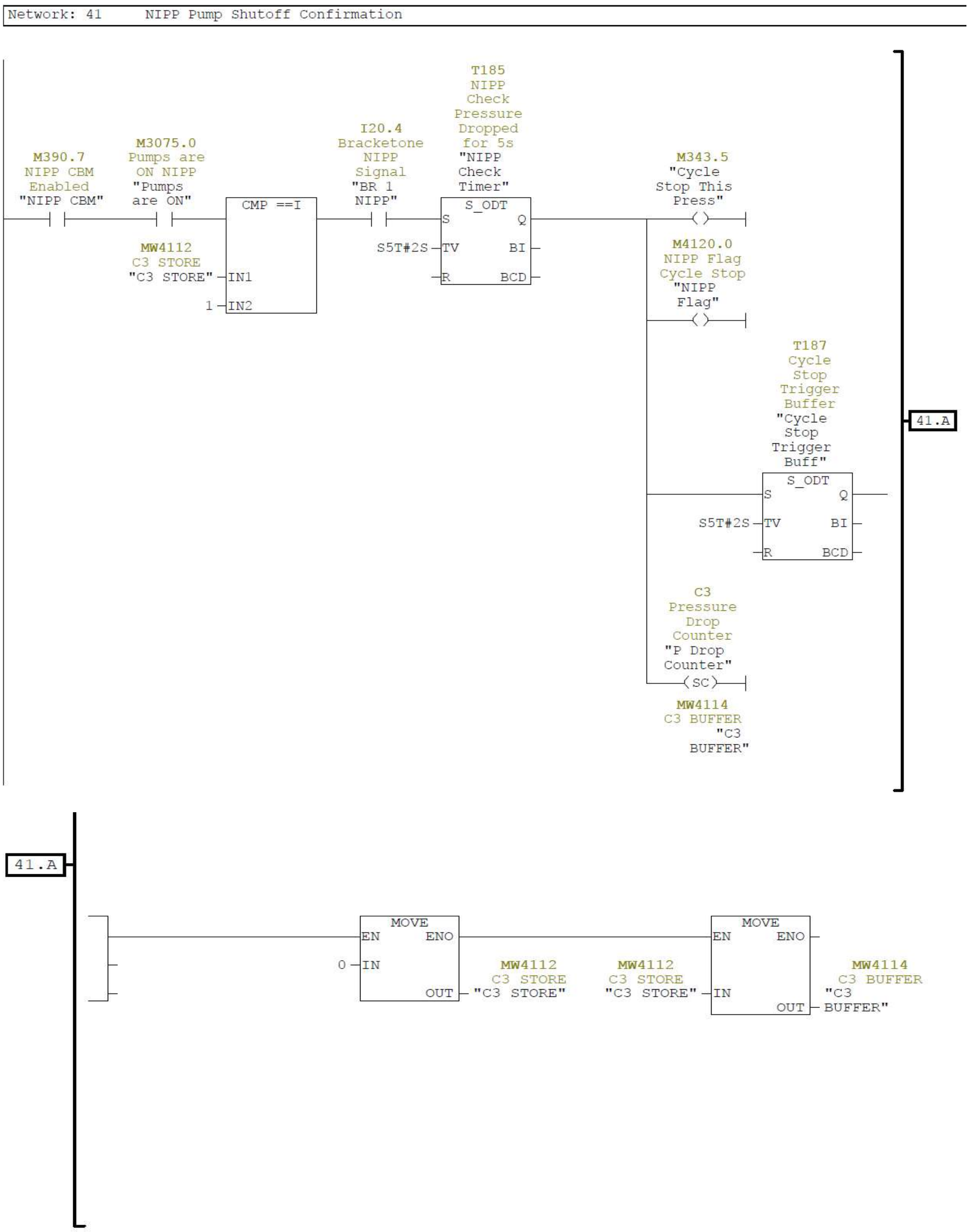


Figure 3.34: PLC Program NW41

The final two networks (Figure 3.35) provided confirmation of pump shut off and updated a variable to the most recent value. Network 42 confirmed that the press cycle stopped, the pumps were still on, the CM cycle stop flag was activated, waited half a second and then activated an alarm before the pumps were turned off: all in the 2 second window mentioned in network 41. Network 43 was scanned at the end of every PLC cycle which ensured it was up-to-date and it was programmed to move the most recent ‘pressure bracket number’ into ‘MW4108’.

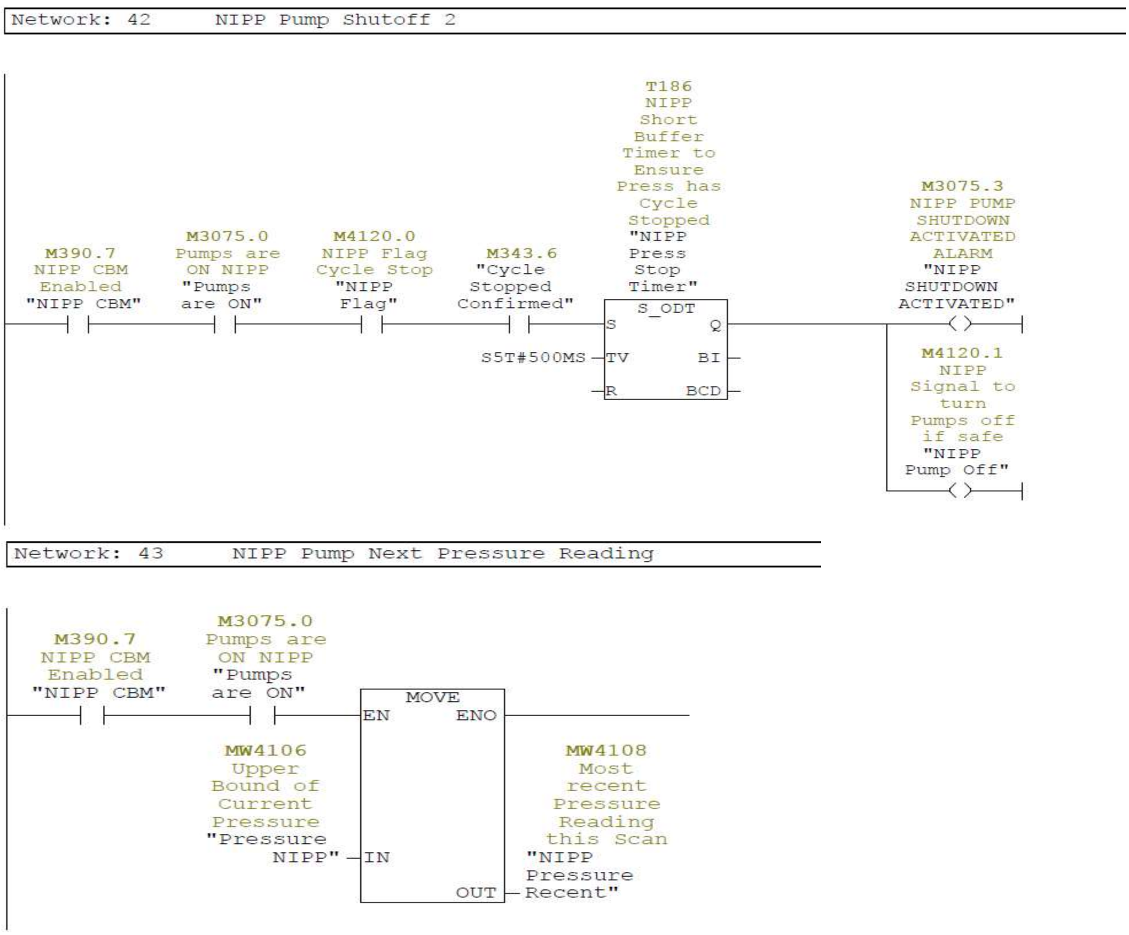


Figure 3.35: PLC Program NW42-43

The system contained adjustable variables which ensured the best CM setup was achieved.

3.8 Summary

This chapter stated the aims and objectives of each individual sensor and the control system as a whole with designs devised and discussed. The next chapter contained the initial simulations and tests performed and their findings.

4 Simulations, Prototypes, Results and Analysis

4.1 Introduction 4-64

4.2 Clutch and Brake Sensor 4-64

4.3 Vibration Analysis Sensor 4-68

4.4 Non-Intrusive Pipe Pressure Sensor 4-73

4.5 Power, Interfacing Circuits, PLC Program and Wireless Transmission 4-75

4.6 Summary 4-78

4.1 Introduction

The initial circuit designs were tested. The results from the tests were compared with expectations from the theory. Prototypes were built and further testing was carried out on them. The designs were refined based upon the findings of this chapter.

4.2 Clutch and Brake Sensor

4.2.1 Simulations

A series of simulations were performed on this sensor. The program located in Appendix A1 was loaded into UnoArduSim V2.6.0, (Simmons, 2020) to simulate scenarios and functionality. The simulations proved successful which allowed for further testing.

4.2.2 Prototype Testing

Four tests were devised to test the functionality of the C&B sensor system. Simulated signals from the PLC were used to mimic the signals from the machine at different states. Additionally, measurements were analysed for accuracy. The distance between the sensor and the reflective plate was set to 6.5cm which was measured with a digital Vernier calliper.

The first test was the lifebit check. Table 4.1 contained the expected results.

Table 4.1: Lifebit Check C&B Test - Expected Results

Expected Results			
Lifebit MC	**Lifebit PLC**	**PLC OK?**	**MC OK?**
0	0	NO	NO
0	1	NO	YES
1	0	YES	NO
1	1	YES	YES

The only state where both the PLC and microcontroller were predicted to be OK was when both sent out their lifebits to each other. The 2nd test, Table 4.2, checked that an initial calibration measurement was only taken when the press was at TDC and press motive was off.

Table 4.2: PLC Input Calibration Check C&B Test - Expected Results

Expected Results		
Press TDC	**Press Motive**	**Calibration Possible?**
0	0	NO
0	1	NO
1	0	YES
1	1	NO

The only state where calibration was expected to be possible was when the Press TDC signal was high and the Press Motive signal was low. The 3rd check tested the calibration distance measurement. Firstly, the calibration button was not pressed and the measurement was recorded; then 10 measurements were taken from the sensor with the button pressed. The range of measurement, mean and standard deviation were analysed in Table 4.3.

Table 4.3: Initial Measurement C&B Test - Expected Results

Expected Results		
Calibration Button Pushed	**Measured Distance (cm)**	**Calibration Complete Signal**
NO	NOTHING	0
YES	6.50	1
	6.50	
	6.50	
	6.50	
	6.50	
	6.50	
	6.50	
	6.50	
	6.50	
	6.50	
Range (cm)		6.50 – 6.50
Mean (cm)		6.50
Standard Deviation (cm)		0

The expected results were idealistic, in reality variance was expected. Therefore, a more reasonable aim was that the average value was within ± 0.04cm of the required value. The 4th test measured different set differences. All 4 possible states of OK, Wearing, NOK and Limit Exceeded were tested as per Table 4.4.

Table 4.4: General Measurement C&B Test - Expected Results

Expected Results					
Set Difference (cm)	**Average Measured Difference – 54 measurements (cm)**	**OK**	**Wearing**	**NOK**	**Limited Exceeded**
0.04	0.04	Y	N	N	N
0.08	0.08	Y	N	N	N
0.12	0.12	N	Y	N	N
0.16	0.16	N	Y	N	N
0.20	0.20	N	N	Y	N
1.5	1.5	N	N	N	Y

4.2.3 PLC and Sensor Incorporation Test Results

The results from the 4 tests were collated in: Table 4.5, Table 4.6, Table 4.7 and Table 4.8.

Table 4.5: Lifebit Check C&B Test Results

Test 1 Results			
Lifebit MC	**Lifebit PLC**	**PLC OK?**	**MC OK?**
0	0	NO	NO
0	1	NO	YES
1	0	YES	NO
1	1	YES	YES

As expected, the system stopped working on either side when a lifebit was not received.

Table 4.6: PLC Input Calibration Check C&B Test Results

Test 2 Results		
Press TDC	**Press Motive**	**Calibration Possible?**
0	0	NO
0	1	NO
1	0	YES
1	1	NO

The interfacing circuits performed as expected. The programming was proven by not allowing calibration to take place until the press was at TDC and there was no motive signal. The system was setup with a measurable distance of 6.50cm and Equation 4.1 was used (identical to Equation 3.3 but with different labels):

$$Distance = \frac{13.147}{Voltage * 0.0048828125} - 0.42 \qquad (\text{Eq. } 4.1)$$

Table 4.7 showed the results.

Table 4.7: Initial Measurement C&B Test Results

Test 3 Results		
Calibration Button Pushed	**Measured Distance (cm)**	**Calibration Complete Signal**
NO	NOTHING	0
YES	6.50	1
	6.51	
	6.50	
	6.50	
	6.50	
	6.48	
	6.49	
	6.50	
	6.51	
	6.50	
Range (cm)		6.48 – 6.51
Mean (cm)		6.499
Standard Deviation (cm)		0.0083

The mean was within +-0.001cm of the correct value whereas the range spanned 0.03cm. The low standard deviation from the mean proved that the averaging method devised in the programs was beneficial in reducing the effects of interferences which potentially caused the different readings. The final test checked distance measurements from the calibration point; the reflective piece was moved a set distance – measured by the callipers – and then 54 measurements were taken. The readings were averaged and then checked that they were within the correct range, the results were added to Table 4.8.

Table 4.8: General Measurement C&B Test Results

Test 4 Results					
Set Difference (cm)	**Average Measured Difference – 54 measurements (cm)**	**OK**	**Wearing**	**NOK**	**Limited Exceeded**
0.04	0.0441	Y	N	N	N
0.08	0.0737	Y	N	N	N
0.12	0.131	N	Y	N	N
0.16	0.157	N	Y	N	N
0.20	0.190	N	N	Y	N
1.5	1.480	N	N	N	Y

The summary of results showed that the average measurements fell within the correct ranges. However, larger deviations were apparent from the 54-measurement average than from the 10-measurement average; therefore, the number of measurements before averaging was kept at 10. Most of the averages read lower than their actual value but because the brackets were relatively wide there were not problems in this test. A series of graphs were produced in Appendix A17 with all 54 data points on to highlight the variance of the signal.

4.3 Vibration Analysis Sensor

4.3.1 Program Tests and Simulations

The first test measured when there was no vibration – the accelerometer was mounted on a breadboard and left still. The second test mounted the accelerometer to a DC motor and fan rotating at 2400 RPM: there was slight mechanical looseness between the fan and motor. The aim of the tests was to see if the program processed the accelerometer data correctly and showed clear spikes at the fundamental frequency and harmonics where appropriate.

4.3.2 Steady State Test Results

With no direct vibration measured the peaks of the FFT graphs were predicted to be low; however, noise and general background vibration was detected. The graphs were windowed to 500Hz. Figure 4.1 showed the result.

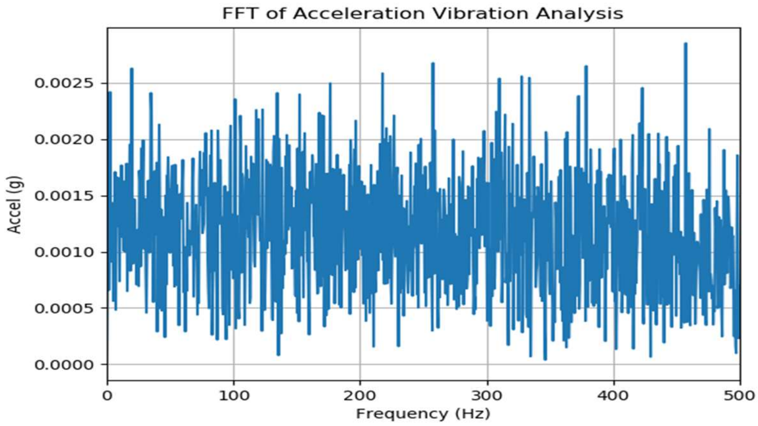


Figure 4.1: Steady State Test Results Acceleration FFT Graph

It was clear that there was no noticeable peak for any frequency. The amplitude was noted to be very low and represented the expected results well; the highest acceleration was less than 0.0030g. Figure 4.2 showed the result of the interference removing methods used to obtain the velocity FFT.

.

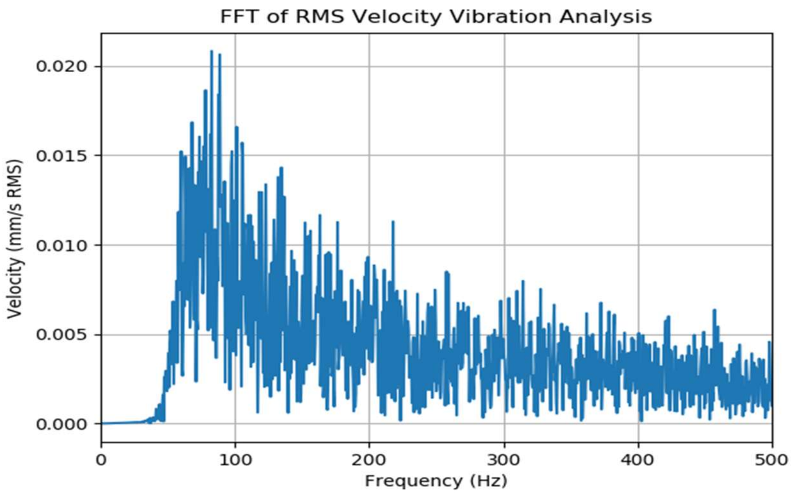


Figure 4.2: Steady State Test Results Velocity RMS FFT Graph

The graph was not as clear as the acceleration FFT graph. This difference in clarity was attributed to the inherent issues with the amplification of errors during numerical integration. The effect of the filter was clear at the start of the frequency range. Nevertheless, the amplitude peaked at less than 0.025mm/s RMS which was reassuring.

4.3.3 2400 RPM Motor Test Results

The accelerometer measured the non-drive end (NDE) of a 2400 RPM motor. The first trace in Figure 4.3 showed the acceleration data in the time domain. All of the graphs in this test used the z-axis. Where a graph contained two plots, the blue trace showed data before the DC offset was removed whereas the orange plot was after it was removed.

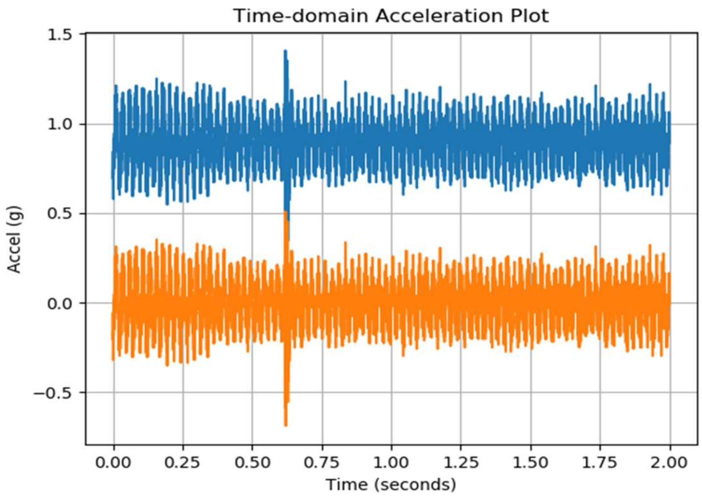


Figure 4.3: 2400 RPM Test Results Acceleration v Time Graph

The graph showed the periodic data over 2 seconds. Initially the data was centred around 1g – because due to the gravitational pull of the earth the z-axis measured 1g at rest. The removal of the DC offset eliminated this error; the accelerometer was successfully calibrated in this manner. The data was then integrated to give Figure 4.4.

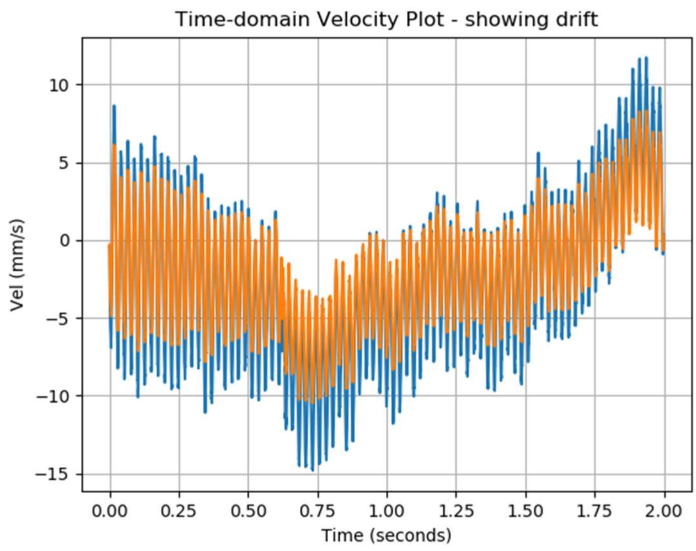


Figure 4.4: 2400 RPM Test Results Velocity v Time Graph

There was a definitive drift in the dataset that needed correcting. Figure 4.5 showed the effect of detrending the dataset and replotting it.

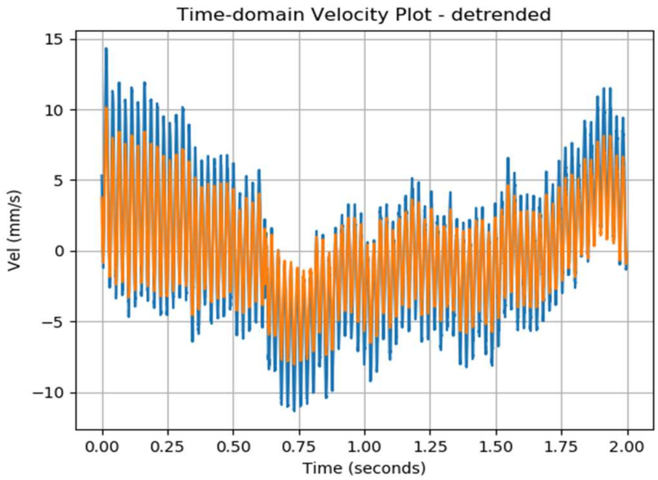


Figure 4.5: 2400 RPM Test Results Detrended Velocity v Time Graph

The signals shifted to centre around 0mm/s and the general trend was more linear. However, it was still not optimum for processing with the FFT. The high pass filter – of 20Hz - was applied to the DC-offset removed detrended data. Figure 4.6 showed the mm/s velocity and Figure 4.7 showed RMS velocity with the DC offset removed.

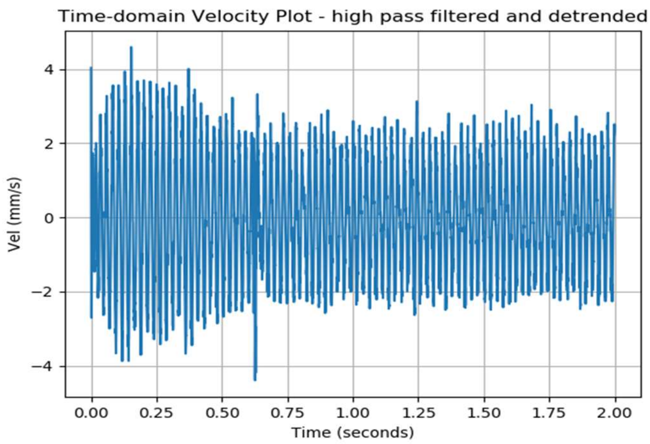


Figure 4.6: 2400 RPM Test Results Detrended & HP Filtered Velocity v Time

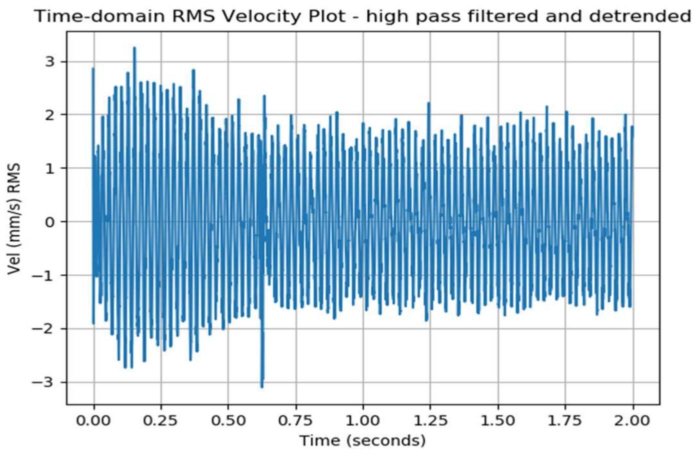


Figure 4.7: 2400 RPM Test Results Detrended & HP Filtered RMS Velocity v Time Graph

The additional high pass filtering was very effective. The signal showed the same peak between 0.50s and 0.75s like the acceleration data. The RMS plot showed a reduction in amplitude as expected. The acceleration and RMS velocity datasets were put through the FFT. Figure 4.8 showed the acceleration FFT graph.

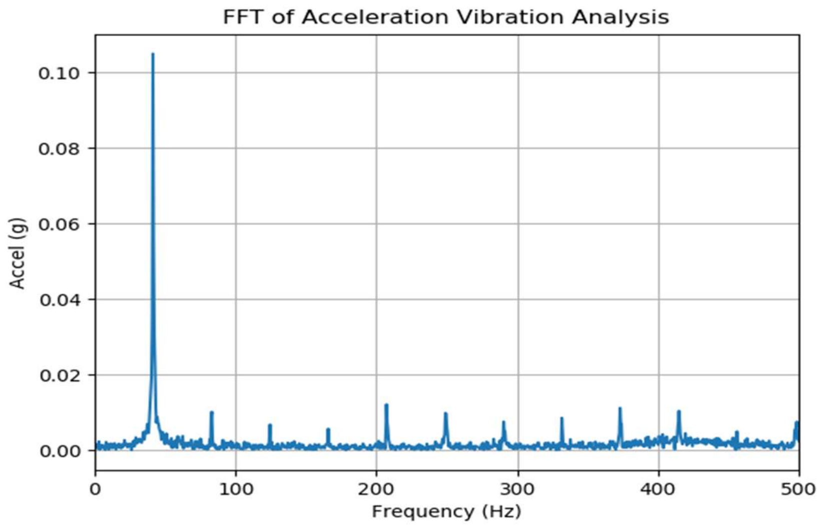


Figure 4.8: 2400 RPM Test Results Acceleration FFT Graph

There was a large initial peak followed by subsequent harmonic frequency peaks. This motor was rotating at 2400 RPM at a fundamental frequency of 40Hz (2400/60) and the detection of this frequency was proof of functionality of the system. The software detected the first 3 peaks – the number was expandable where required - at: 41.344Hz with 0.105g; 83.191Hz with 0.01g and 124.535Hz with 0.007g. The primary peak was 1.344Hz above the theoretical position and the harmonics were close to 2n and 3n, where n was 41.344Hz.

As mentioned previously slight mechanical looseness was physically identified and the software had to see 3 harmonics with amplitudes of at least 20% of the 1st harmonics peak (0.021g). All harmonics were under the value at the time of measurement and so the program output an 'OK' signal. As a test of program functionality, the tolerance of the harmonic peak percentage for mechanical looseness was changed from 20% to 5%. Subsequently the wearing condition was triggered and pointed to mechanical looseness specifically. Figure 4.9 showed the FFT graph of the RMS velocity data.

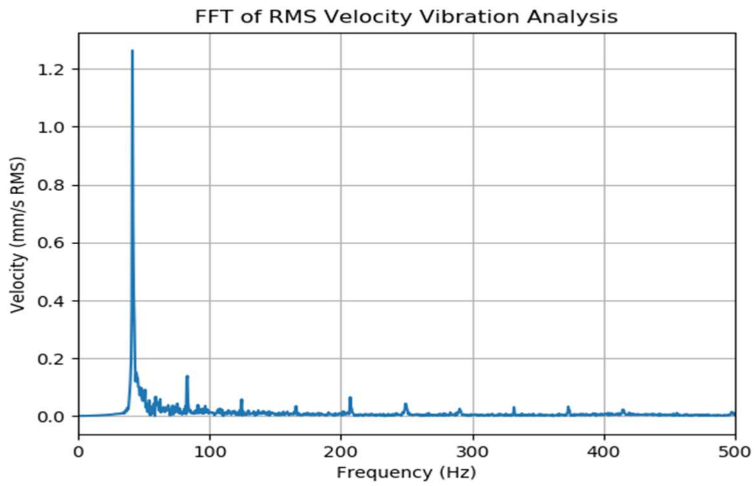


Figure 4.9: 2400 RPM Test Results RMS Velocity FFT Graph

The software detected the first 3 peaks at: 41.344Hz with 1.264mm/s RMS; 83.191Hz with 0.139mm/s RMS and 124.535Hz with 0.058mm/s RMS. The peak positions followed those of the acceleration graph and the conclusions drawn were the same.

4.4 Non-Intrusive Pipe Pressure Sensor

4.4.1 Simulations

A series of simulations were performed on this sensor. The Arduino program located in Appendix A6 was loaded into UnoArduSim V2.6.0, (Simmons, 2020) to ensure the program worked as expected. The functionality was proven and due to the standardisation between the Arduino programs it was easier to achieve.

4.4.2 Strain Gauge and Wheatstone Bridge Simulations

The Wheatstone bridge featuring the two strain gauges was designed on Multisim simulation software using the tolerances identified in the research. Figure 4.10 showed the circuit setup.

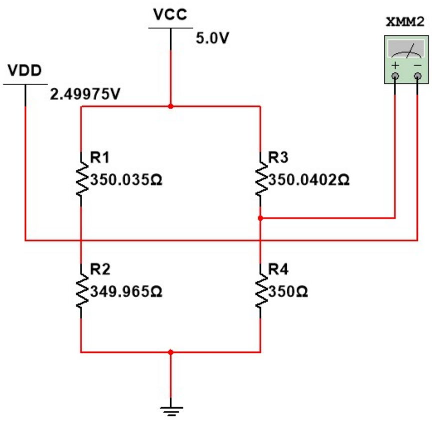


Figure 4.10: Strain Gauge and Bridge Circuit

The potential divider on the left took the values from Table 3.13 of the resistor's tolerances. The two strain gauges were replaced by fixed resistors of theoretical resistance values at set pressures; the setup seen in Figure 4.10 was for 80bar pressure. An additional power supply was required because the software could not differentiate between 2.5V and the 2.49975V coming from the potential divider. The multimeter was setup to read the difference between the dividers, with R3 changing with pressure and R4 holding at 350Ω due to the assumption of no significant resistance change. The result was showed in Figure 4.11.

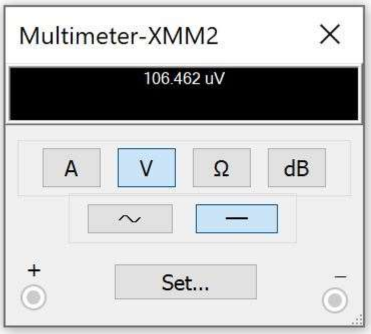


Figure 4.11: Strain Gauge and Bridge Simulations

This result of 106.462μV was the same as the theoretical calculations. All of the other pressures were tested with different resistance values and the voltages matched those calculated. The next part of the system was amplification. Unfortunately, the INA125 IA was not available in the electronic simulation software used so it was decided to test the IA physically.

4.4.3 Prototype Testing

A prototype was constructed for testing purposes. The strain gauges and IA amplification were tested by connecting an oscilloscope to the output where the microcontroller was planned to connect. Pressure was applied to the active strain gauge mimicking the pressure application from the pipe. The gain was adjusted so that a clear signal was visible on the oscilloscope; different gains were used when the microcontroller was connected in the final test. An oscilloscope trace in Figure 4.12 showed a varying voltage as pressure was applied to the gauge.

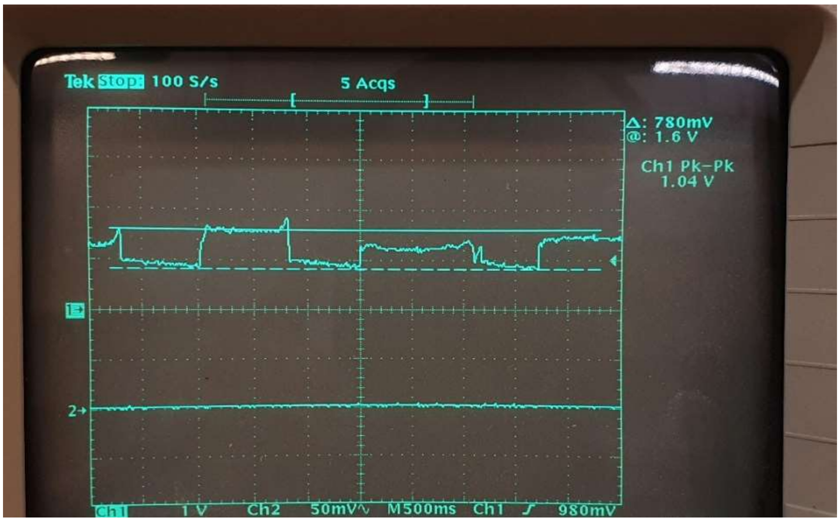


Figure 4.12: Prototype Test of NIPP Sensor

The signal showed clear variation in voltage amplitude that directly corresponded to pressure applied. There was limited noise on the signal when at a steady state which proved the effectiveness of the IA in removing sources on interference which would have otherwise affected the CM findings. The prototype design was proven to be functional and full integration with the interfacing circuitry and microcontroller was undertaken for the full testing stage.

4.5 Power, Interfacing Circuits, PLC Program and Wireless Transmission

4.5.1 Power to the Sensors

Each of the sensors powered up correctly and all functions operated satisfactorily; the tests included transmission via the nRF module and the operation of I/O. The portable powerpack worked well with the CGD and the VA's DC-DC converter powered the sensor and screen properly.

4.5.2 24V Digital PLC Signal to 5V/3.3V Digital Microcontroller Signal

This simple circuit was simulated and tested physically. Figure 4.13 showed the circuit and its multimeter reading. The theoretical result of 4.225V was replicated in the simulation.

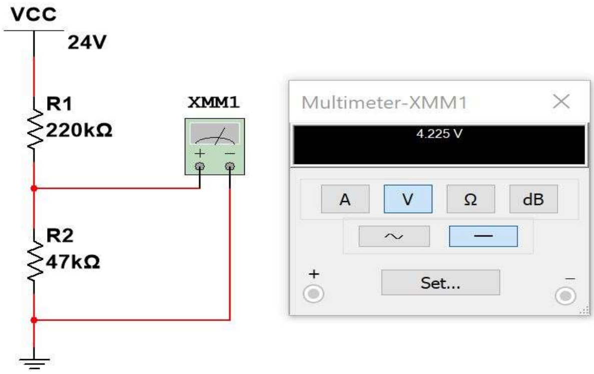


Figure 4.13: 24V to 5V Divider Simulation Results

Testing the resistor circuit practically gave a voltage of 4.21V; a slight drop but still acceptable. The 3.3V variant (Figure 4.14) gave the following result:

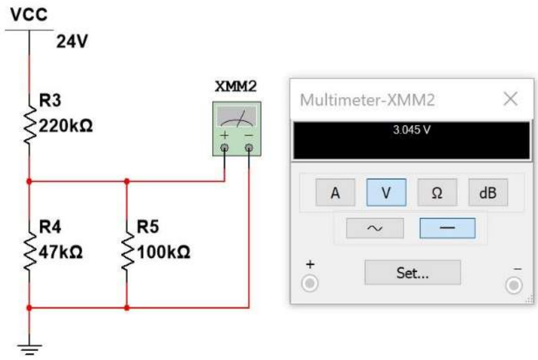


Figure 4.14: 24V to 3.3V Divider Simulation Results

The simulated voltage was the same as the theoretical. The measured voltage was also slightly lower at 3.02V. The drops were caused by the tolerances of the physical resistor.

4.5.3 5V/3.3V Digital Microcontroller Signal to 24V Digital PLC Signal

The circuits were built in the Multisim simulation software and tested at two points. The first meter tested the first potential divider and the second test point was at the output of the optocoupler to see if it switched 24V. Figure 4.15 showed the 5V circuit and the simulation results.

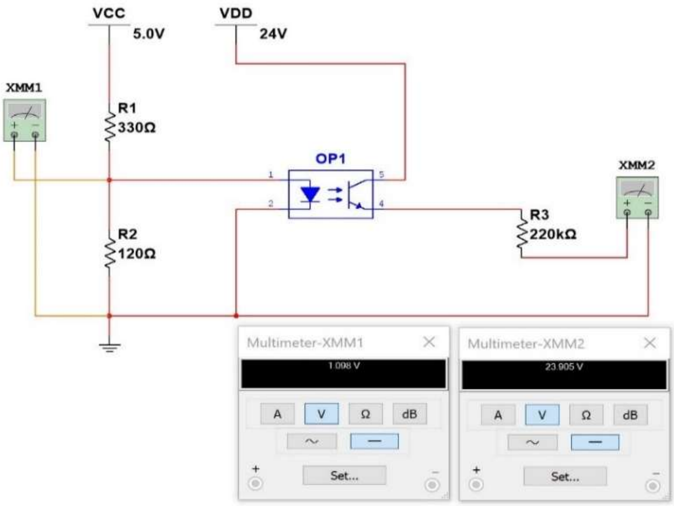


Figure 4.15: Optocoupler 5V Simulation Results

The theoretical value for the voltage after the first divider was 1.33V. The simulation showed 1.098V which was 0.232V lower than expected. The voltage was still able to switch the optocoupler because, the output voltage was 23.905V. The 3.3V simulation changed the bottom resistor value and the input voltage to the potential divider, Figure 4.16 showed that the results were as expected.

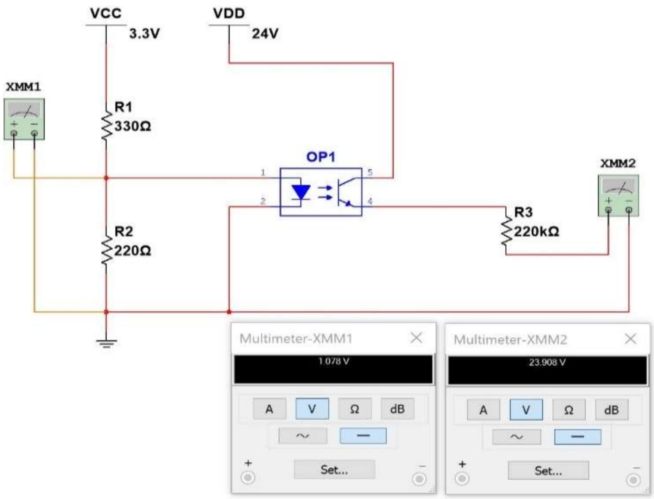


Figure 4.16: Optocoupler 3.3V Simulation Results

The prototype constructions of both these optocoupler circuits successfully switched 24V at their outputs.

4.5.4 PLC Program

For logistical reasons the PLC program was tested with the circuitry connected to it when the final results were gathered in Chapter 5.

4.5.5 Wireless Transmission and Central Graphical Device

All microcontrollers were setup with their own nRF transmitters as per Figure 3.16 and the microcontrollers were loaded with their test programs. The signals sent were varied to test possible combinations and to see if they successfully transmitted to the CGD. The main test was for the C&B sensor to send a NOK signal, the VA sensor to send an OK signal and the NIPP sensor to send a 06 signal. The individual strings were ‘1NO’; ‘2OK’ and ‘306’. Figure 4.17 showed the Python Shell which printed the data it received.

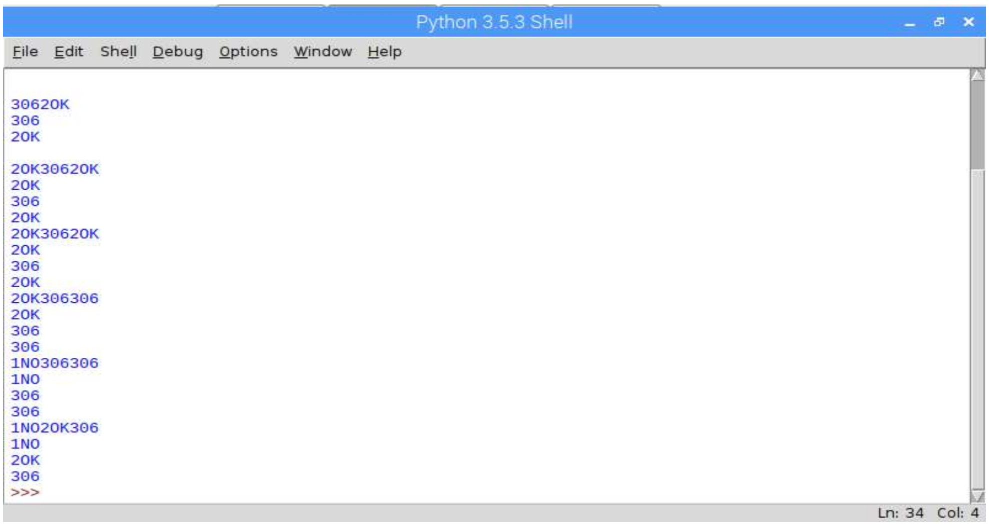


Figure 4.17: nRF and GUI 1st Test Python Shell

The results showed that it took a brief period of time before all 3 distinct data strings were read. Multiple readings where all sensors were not represented did not generate a GUI which proved that the sorting section of the program was working. Figure 4.18 showed the generated GUI once all sensors reported in.

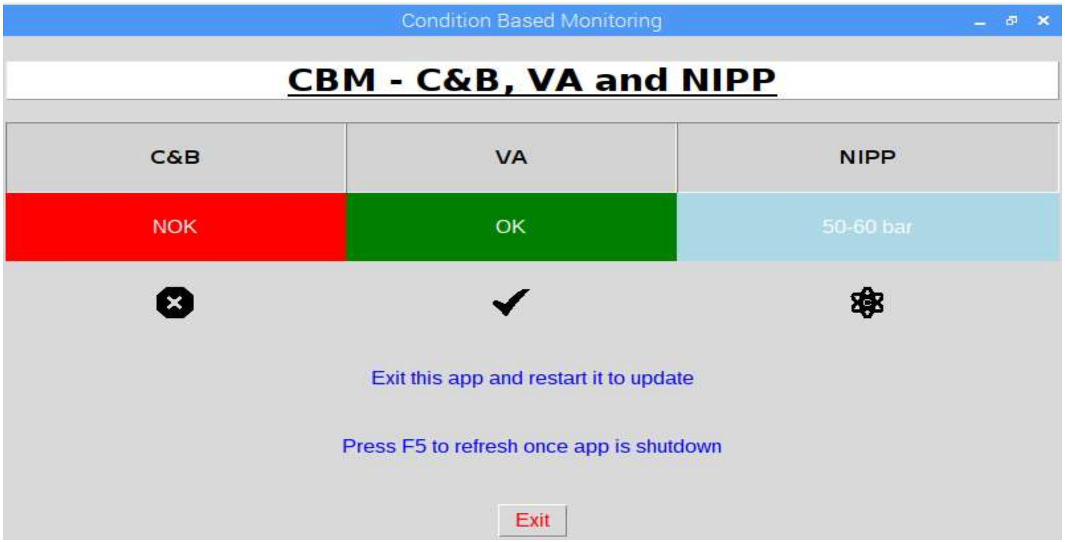


Figure 4.18: Generated GUI from 1st nRF Test '1NO2OK306'

The GUI showed a clear visual representation of the data sent from the microcontrollers with the correct conditions of each sensor alongside text, background colours and icons.

4.6 Summary

The first simulations and prototype builds proved themselves successful. Further testing was scheduled to reaffirm the initial findings and provide reassurance of correct operation on the specific equipment. Improvements were made to the initial designs throughout to enable them to be as accurate as possible.

5 Implementation, Testing and Results

5.1 Introduction 5-80
5.2 Connections Junction Box 5-80
5.3 Clutch and Brake Sensor 5-81
5.4 Vibration Analysis Sensor 5-84
5.5 Non-Intrusive Pipe Pressure Sensor 5-87
5.6 Power and PLC Interfacing 5-91
5.7 Wireless Transmission and Central Graphical Device 5-98
5.8 Summary 5-99

5.1 Introduction

This chapter discussed the implementation of the circuits. The circuit designs were finalised before being soldered to stripboard. After functionality testing, they were put inside enclosures. An additional junction box was designed to act as the connector between the sensors and the PLC. The results gathered in this stage of further testing were analysed alongside the previous findings.

5.2 Connections Junction Box

A metal junction box was used to house terminal rails, an MCB, a switch and two indicator lamps as showed in Figure 5.1.

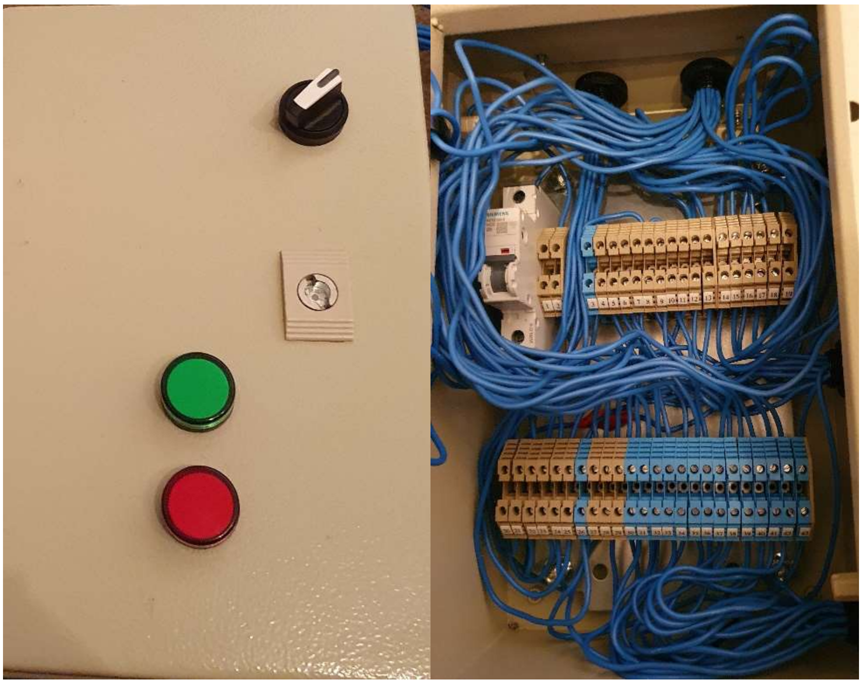


Figure 5.1: Connections Junction Box

The MCB was added as a safety feature for protection. The glands on the top of the box went to the PLC and the glands on the side connected to the sensors via harting plugs. The main switch on the front controlled power delivery to the sensors, the green light indicated power active and the red light was wired in as a spare for any purpose. The schematics created for this box, with label identification of where each terminal goes to and from was attached to Appendix A18. The connections junction box powered up and operated as expected during all testing periods.

5.3 Clutch and Brake Sensor

The C&B sensor circuitry was placed inside an IP rated enclosure as showed in Figure 5.2. The wires from the left gland went to a harting plug for the connection junction box and the wires in the right gland went to the measurement sensor. A terminal designation drawing for the main board was included as Appendix A19.

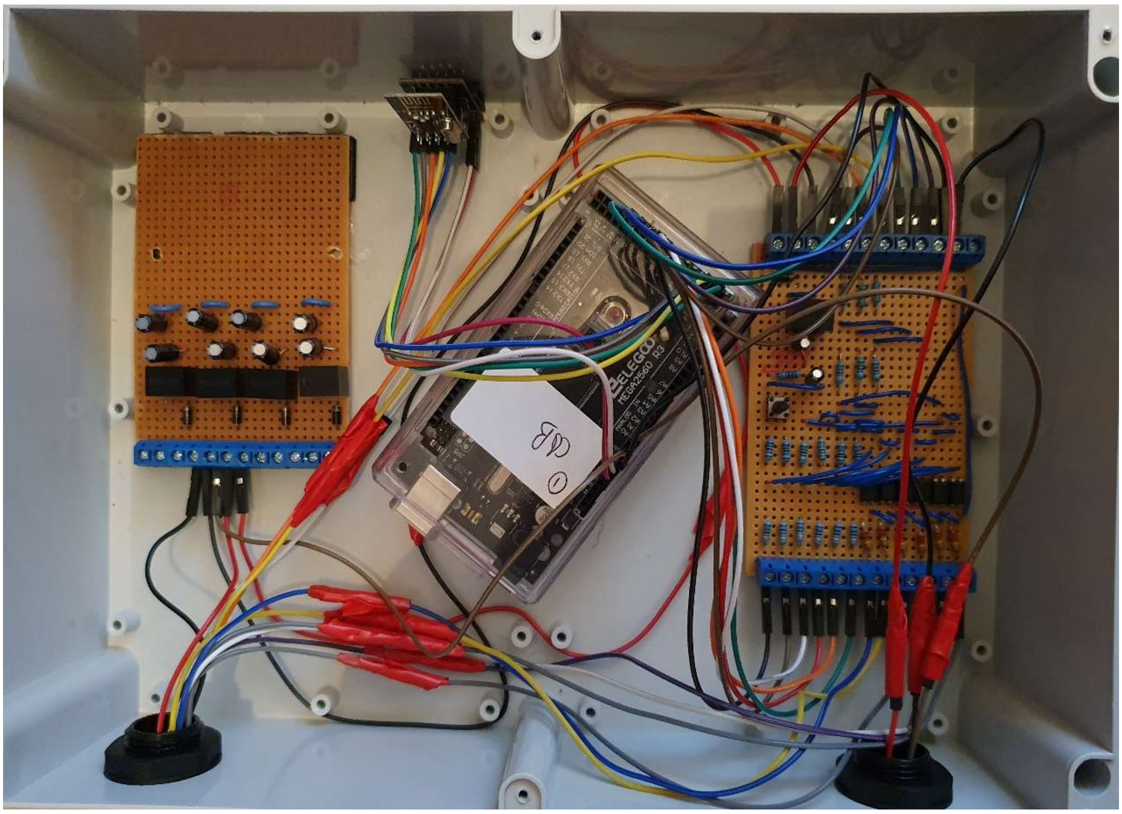


Figure 5.2: Complete Clutch and Brake Sensor Circuitry

The sensor and reflective white sheet were placed on the Clutch and Brake unit as showed in Figure 5.3. The distance between them was 6.5cm. Two additional tests were undertaken: the first test monitored measurements over a small period of time to check stability and the second test used an offset to check condition.

Figure 5.3: Sensor Mounted to C&B unit

The complete program for this C&B sensor was added to Appendix A20. The circuit was calibrated and measurements were taken throughout press cycles; the delays in the program ensured one set of 10 measurements was taken per press cycle. The results from this test were added to Appendix A21. A summary of the 300 measurements taken was drafted into Table 5.1.

Table 5.1: C&B Sensor Implementation Test 1

Set of 10 Readings	Average Difference (cm)	Condition	Press Cycle
1	0.00	OK	1
2	0.00		2
3	0.01		3
4	-0.01	N/A	4
5	-0.01		5
6	0.00		6
7	0.00	OK	7
8	0.01		8
9	0.02		9
10	0.02	OK	10
11	0.01		11
12	0.00		12
13	0.00	N/A	13
14	0.00		14
15	**-0.01**		15
16	0.00	OK	16
17	0.00		17
18	0.00		18
19	0.00	OK	19
20	0.01		20
21	0.01		21
22	0.01	OK	22
23	0.00		23
24	0.00		24
25	**0.03**	OK	25
26	0.01		26
27	0.00		27
28	0.00	OK	28
29	0.00		29
30	0.01		30

The average readings varied from -0.01cm to 0.03cm; a 0.04cm range. The measurements were taken at the correct times when the press was stopped at TDC but there was still some vibration present in the area due to the rotating flywheel, this accounted for the slightly larger range. Despite this, out of 10 averages the correct condition was reported 8 times and the other 2 times no condition was reported. Therefore, there were no incorrect condition reports sent to the PLC or CGD.

The first test was conducted on a unit with new pads and as such there was no wear expected and no wear was found. A different unit was also tested that had worn pads – a recent check using the old method found that the pads had worn by 0.13cm from new. The sensor was designed to be installed on new pads only so to measure the true wear on this setup the measured value had 0.13cm loss added to it; this test was completed to prove the sensor could operate in this closer range. The complete graph was added to Appendix A22. Table 5.2 provided a summary of the results – condition 'WE' was 'Wearing'.

Table 5.2: C&B Sensor Implementation Test 2

Set of 10 Readings	Average Difference (cm)	Condition	Press Cycle
1	0.13		1
2	0.14	WE	2
3	0.14		3
4	0.14		4
5	0.13	WE	5
6	0.14		6
7	0.14		7
8	0.13	WE	8
9	0.15		9
10	0.13		10
11	0.14	WE	11
12	0.14		12
13	0.13		13
14	0.14	WE	14
15	0.13		15
16	0.13		16
17	0.14	WE	17
18	0.15		18
19	0.13		19
20	0.14	WE	20
21	0.14		21
22	0.15		22
23	0.13	WE	23
24	0.15		24
25	0.13		25
26	0.13	WE	26
27	0.12		27
28	0.13		28
29	0.14	WE	29
30	0.14		30

The correct condition was reported across all averages. The individual values varied a lot, which further reinforced the necessity of using averages and obtaining 3 consecutive identical results before confirming the condition to the PLC and CGD. The accuracy in condition reporting was better when the measurement was not close to a boundary value.

5.4 Vibration Analysis Sensor

A photograph of the circuit and a terminal designation was added to Appendix A23. The complete program for this sensor remained fundamentally unchanged from Appendix A4. However, during testing alterations were made to the filter, windowing and condition monitoring section. The Arduino which controlled the nRF function of this sensor used the same program as found in Appendix A13. The circuit in the enclosure was showed in Figure 5.4.

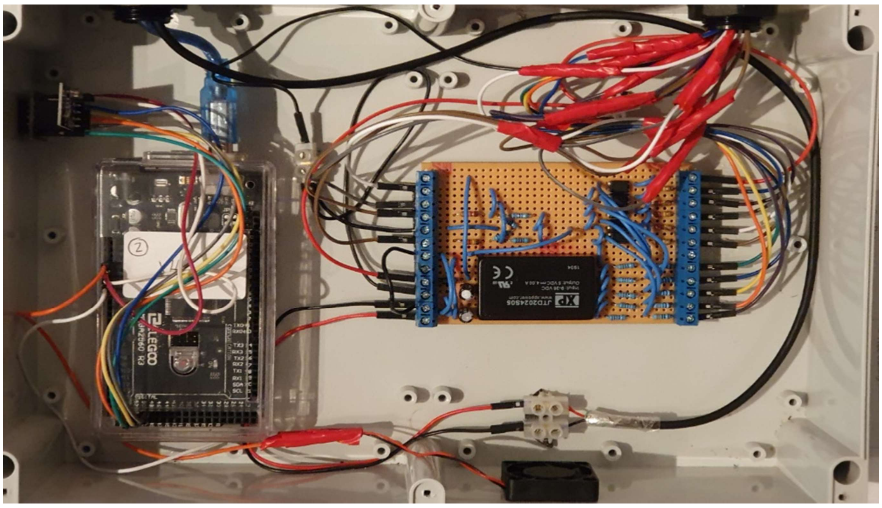


Figure 5.4: VA Circuit Boards

Two further VA tests were completed. The first looked at a main motor which had recently been maintained (Figure 5.5). The accelerometer was fastened to the fixture which was installed for taking VA measurements on the NDE of the motor; electrical tape was also added to cover the electronics surface.

Figure 5.5: Main Motor VA

From historical records the acceleration was most frequently analysed for this motor and so the FFT acceleration graph was produced as showed in Figure 5.6.

.

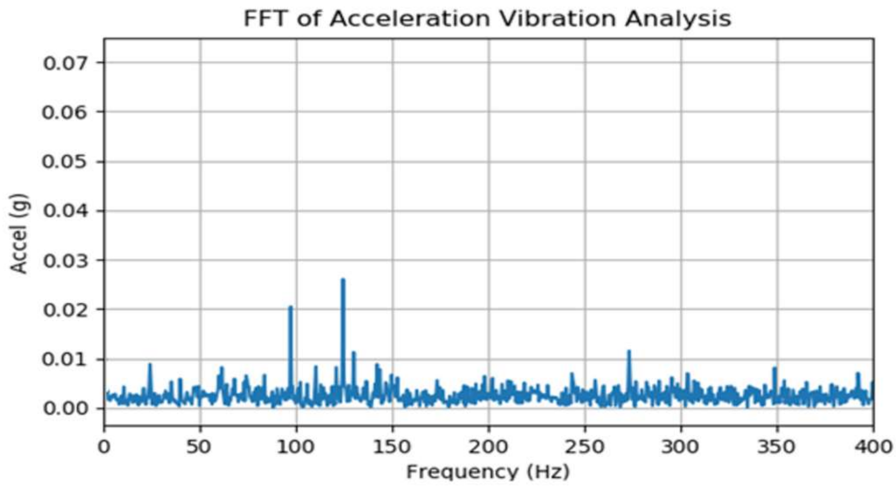


Figure 5.6: Main Motor Acceleration FFT

This motor ran at a fixed 1488 RPM and the fundamental frequency was 24.8Hz. The peak detection identified peaks at 25Hz, 98Hz, 123Hz and 270Hz: these were the fundamental frequency and the 4th, 5th and 11th harmonic. These peaks exceeded the amplitude of the fundamental frequency by a considerable amount and the condition monitoring identified mechanical looseness. Considering the motor had just undergone maintenance this was plausible and further work was planned in to secure all fastenings. The amplitude was relatively low because the motor was not under a lot of strain and the positioning of the accelerometer mount reduced amplitude; the relative differences between harmonics was sufficient for this analysis.

The second test was performed on a small motor used in die clamps which ran at 6000 RPM, the fundamental frequency was 100Hz. Upon physical inspection there was no obvious issue with the clamp and so it was operated and tested. Figure 5.7 showed the acceleration FFT graph from the data. There was a peak at 103Hz of 0.026g which aligned with the fundamental frequency and it was not abnormally large in amplitude so there was no unbalance.

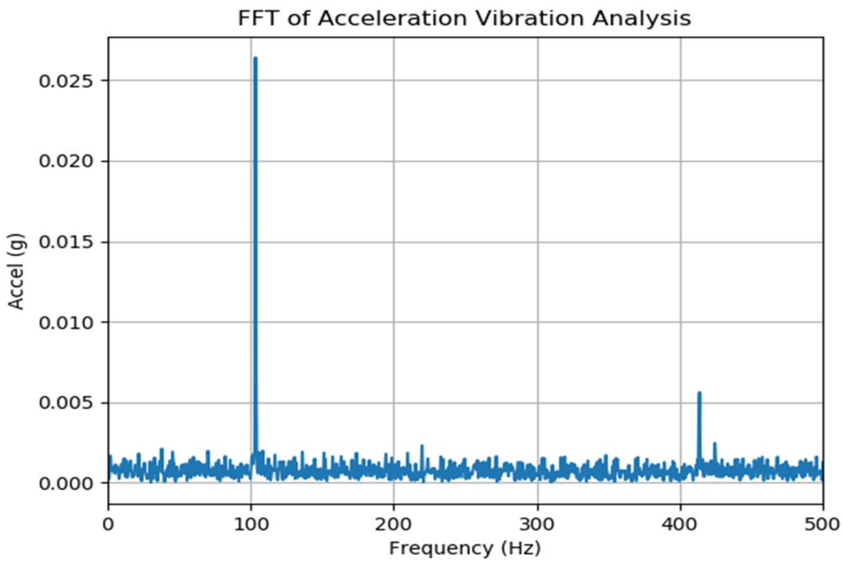


Figure 5.7: 6000 RPM Test Results Acceleration FFT Graph

The peak at 417Hz was less than a fifth of the primary peak and so did not indicate misalignment or mechanical looseness and an ‘OK’ CM signal was transmitted. The RMS velocity graph (Figure 5.8) showed the same conclusions, the effect of the filter was clearly visible and the 417Hz peak was smaller.

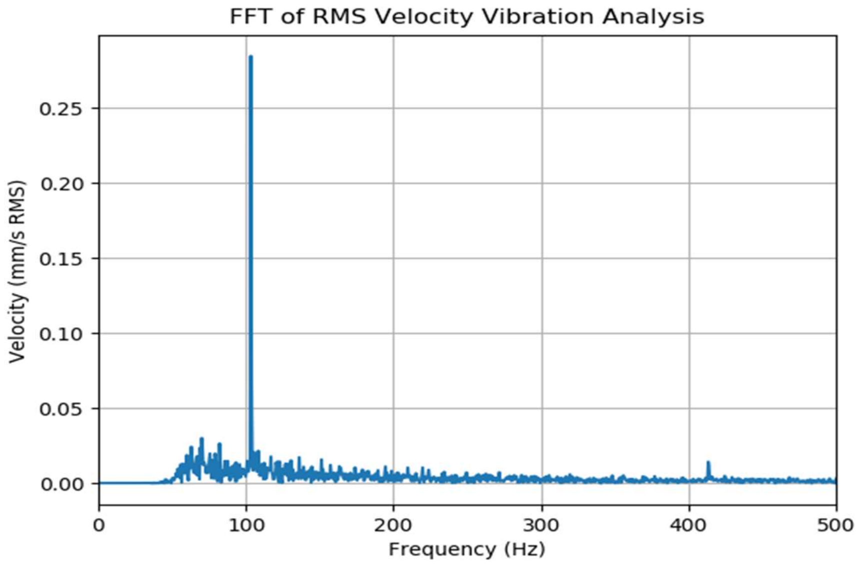


Figure 5.8: 6000 RPM Test Results RMS Velocity Graph

The 4 different tests performed with the VA sensor provided useful data on the conditions of the equipment. It was noticed that the drift caused by integration was not always linear and as such the linear detrend function was not as effective as it could have been in places; although the filtering did help.

5.5 Non-Intrusive Pipe Pressure Sensor

Two circuit boards were constructed for this sensor. The NIPP measuring board layout was added to Appendix A24 and the interfacing circuit was added to Appendix A25. The enclosure that contained the circuit boards was showed in Figure 5.9.

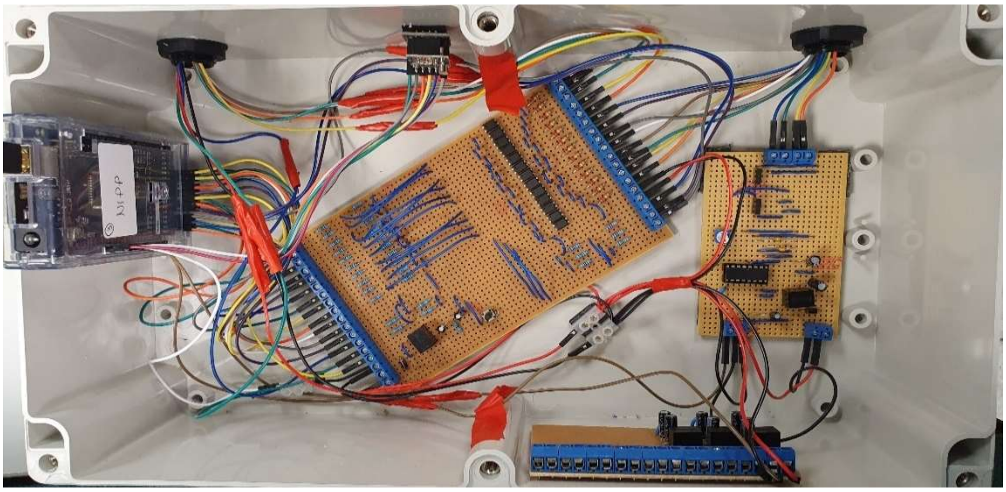


Figure 5.9: NIPP Circuit Boards inside Enclosure

The complete program was placed in Appendix A26. The strain gauges were connected to the hoop and longitudinal faces of the pipe with a thermally sufficient adhesive; as showed in Figure 5.10.

Figure 5.10: Strain Gauge Installation

Several readings were taken over different press cycles and analysis was performed on 2 of the cycles. According to the press settings the pressure never exceeded 65bars on the job which was in operation at the time. The data gathered showed some points exceeded 65bar but the 10-measurement average never exceeded 65bars; these higher pressures were erroneous results. The system was initially calibrated (with gain adjustment) with the pumps off and no pressure inside the pipes: this graph was added to Appendix A27. Table 5.3 included an overview of the data. The average column contained the average of a set of 10 readings and the bracket column contained the determined group bracket number for the 3 average readings, for example ‘01’ represented readings from 0-10bar.

Table 5.3: NIPP 0bar Pressure after Calibration

Number of Reading	Calibration 0bar Pressure (bar)	Average (bar)	Bracket
1	2.71	2.17	01
2	2.71		
3	2.71		
4	2.71		
5	2.71		
6	2.71		
7	2.71		
8	0		
9	2.71		
10	0		
11	2.17	1.85	
12	2.71		
13	0		
14	0		
15	0		
16	0		
17	5.43		
18	0		
19	2.71		
20	5.43		
21	2.71	1.00	
22	1.9		
23	2.71		
24	0		
25	0		
26	0		
27	0		
28	0		
29	0		
30	2.71		

The values varied more so than the C&B system did and this was due to the lower resolution available for this sensor. However, the important average reading remained close to 0bar and well within the ‘01’ bracket. When setting up the gauges it was noted that there was a small amount of strain present when there was no pressure in the pipe, the assumptions made in the theoretical calculations assumed this would not be the case and this explained why the average values were above 0bar. The full data graph was added to Appendix A28 for 2 complete cycles. There was a lot of variation in the signal, sometimes the pressure reading dropped to a negative value which meant that the voltage reading went higher than 0.25V; the cause of this was likely from noise on the signal or vibration in the system causing a variation in the recorded strain. Fortunately, due to other mitigating measures put in place the average readings showed a clear cycle of pressure. An overview of the data was added to Table 5.4.

Table 5.4: NIPP Pressure over 2 cycles

Set of 10 Readings	Average Pressure (bar)	Bracket	Rate of Change
1	0.92	01	0
2	0.60		
3	2.77		
4	15.08	02	1
5	13.27		
6	13.86		
7	22.63	03	1
8	23.06		
9	29.66		
10	45.72	05	1
11	42.48		
12	42.83		
13	17.28	02	3
14	14.05		
15	11.79		
16	2.84	01	2
17	1.27		
18	1.38		
19	0.30	01	1
20	7.60		
21	8.42		
22	46.11	05	0
23	49.88		
24	49.84		
25	29.81	03	2
26	24.55		
27	28.86		
28	2.63	01	3
29	0.79		
30	3.80		

The primary purpose of this sensor was to detect leaks. During the entire testing phase there were no leaks. However, the rate of change was still analysed and where required the rate of change was rounded to an integer. The bracket and rate of change was plotted on Figure 5.11.

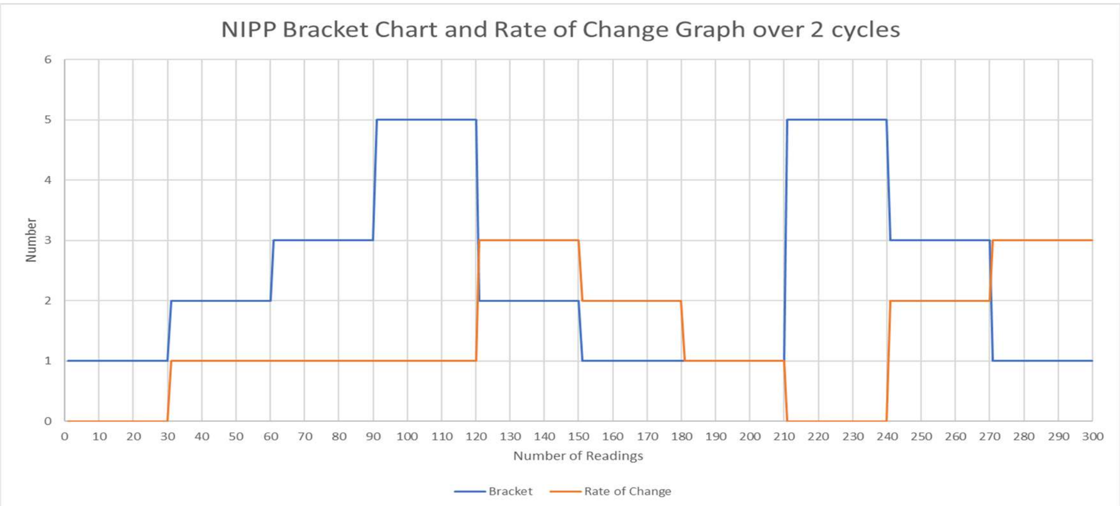


Figure 5.11: NIPP Rate of Change Graph over 2 cycles

The bracket never exceeded 5 while the rate of change between pressures never exceeded 3. The shape of the graph was as expected for 2 normal cycles. The low-resolution bracket graph line showed the pressure output from the sensor as seen by the PLC over time. There was a clear increase in pressure which corresponded to the correct points of the cycle where the press needed to rise and there were clear periods of rest where the press was being unloaded and loaded.

One noticeable impact on the output was the 10-measurement average. While it had the required effect of reducing errors it did not always overlap correctly for the stage of the process it was in. The requirement to have 3 consecutive corresponding averages also proved problematic on occasion. However, the 3 averages helped prevent erroneous misreporting of rates of change of pressures when the press pendulated (a quick short burst of pressure) this was correctly not counted as a leak.

The press cycle varied each time as it normally did due to its mechanical nature, it was decided that in the future experiments with fewer measurements being averaged would be trialled to improve the accuracy and resolution of this sensor. The lead resistance of the gauge wires had a slight effect in altering the resistance and making the microcontroller equation less accurate – further compensation was required. The assumptions made initially, on resistor values and nominal values were found to not always be precise and as such some variation in the final results was attributed to this too; although the laser-engraved resistor pair was precise.

In summary, the sensor showed two clear press cycles by outputting the corresponding brackets and the rate of change never exceeded the threshold for leak indication. The CM aspect of the sensor worked well but due to the resolutions available it was not suitable for high accuracy pressure recordings - which was predicted - hence the choice of 10bar windows. The calibration at 0bar helped to remove some error offsets. Overall, the sensor was suitable for monitoring machine leaks.

5.6 Power and PLC Interfacing

5.6.1 Power

The regulator circuit board photograph and terminal designations were added to Appendix A29. While all 3 sensors were active and transmitting, the total current draw from the main PLC supply was 400mA at 24V; there were no power failures.

5.6.2 PLC Interfacing

All of the connections to and from the PLC read and wrote OK: the PLC input card detected the 24V signals from the sensors and the PLC outputs were stepped down to suitable levels for the microcontrollers to access. There were no problems with grounding – no elevated grounds/potentials and interference and noise were minimised and did not appear to affect the signals.

Networks 1-4 mapped the output lifebit and input lifebits correctly. Networks 5-7 (Figure 5.12) showed the check on the lifebit being complete after 2 seconds had passed.

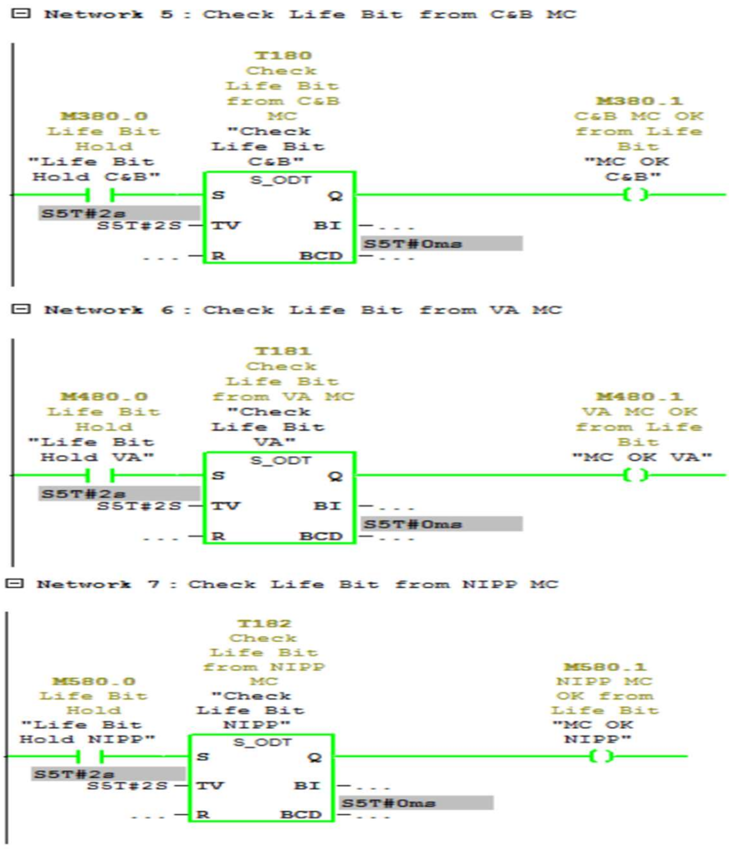


Figure 5.12: PLC Program NW5-7 Online

Networks 8-11 were the enabling networks. Figure 5.13 showed two results of network 8 (the same method was used for networks 9-11) and the 'Auto CBM Enabled' bit was only active once the 'Master On' was toggled in the PLC.

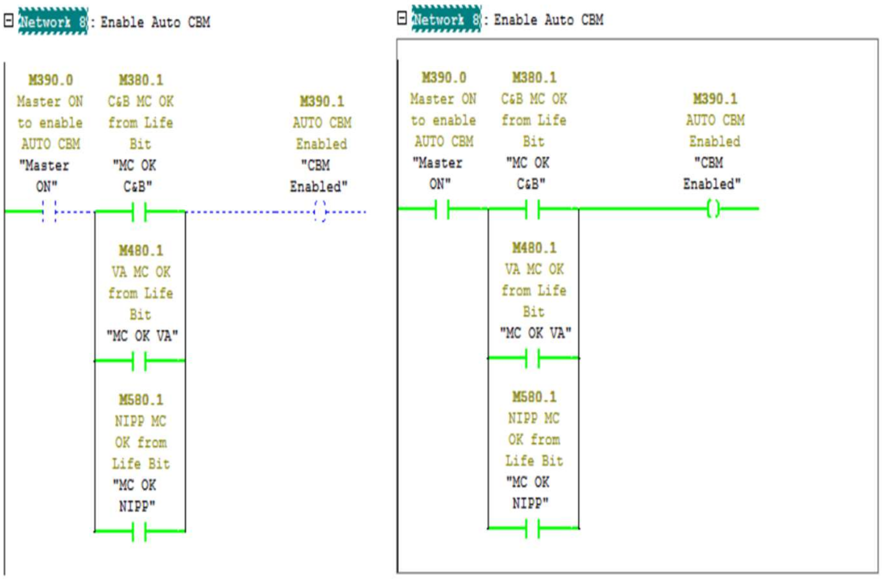


Figure 5.13: PLC Program NW8 Online

Networks 12-15 mapped I/O variables correctly. Network 16 mapped the condition inputs from the microcontroller to memory bits in the PLC. Figure 5.14 showed the C&B wearing condition active with complete preconditions.

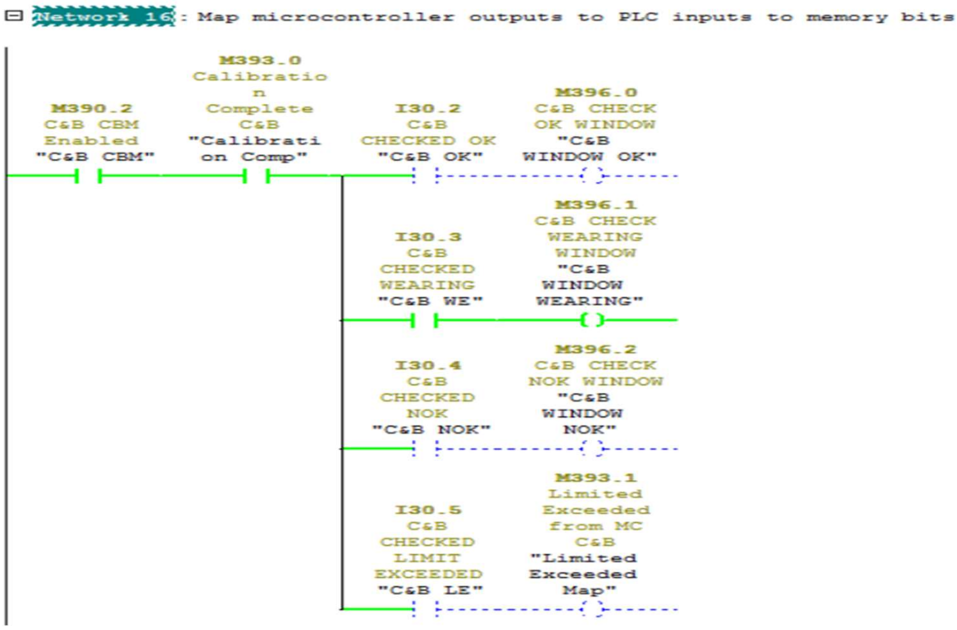


Figure 5.14: PLC Program NW16 Online

The condition memory bits were mapped to their alarm bits in network 17. Network 18 was the first corrective action network. Figure 5.15 showed it without the wearing condition and Figure

5.16 showed it with the wearing condition and its first successful iteration, the reduction in the ram brake position was showed and the counter also increased by 1.

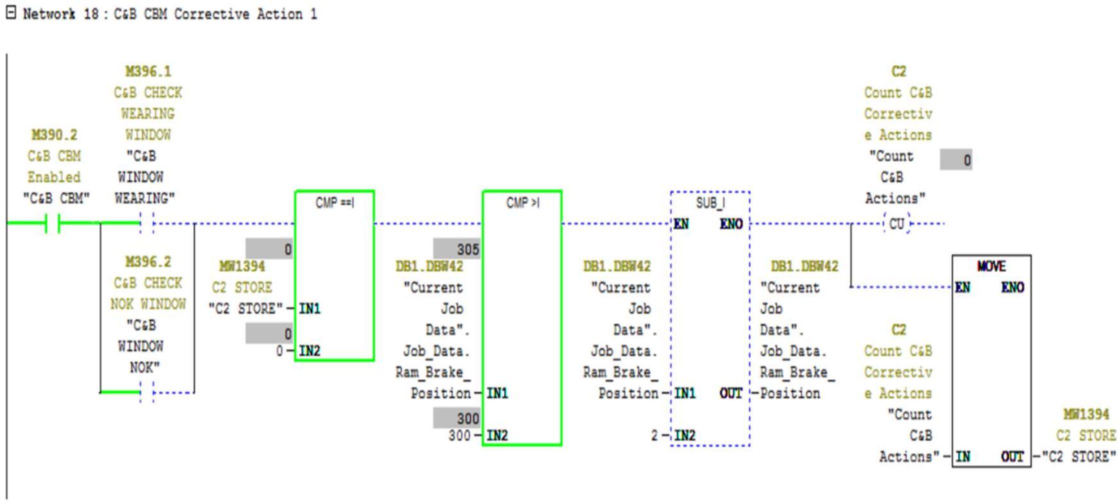


Figure 5.15: PLC Program NW18 Online

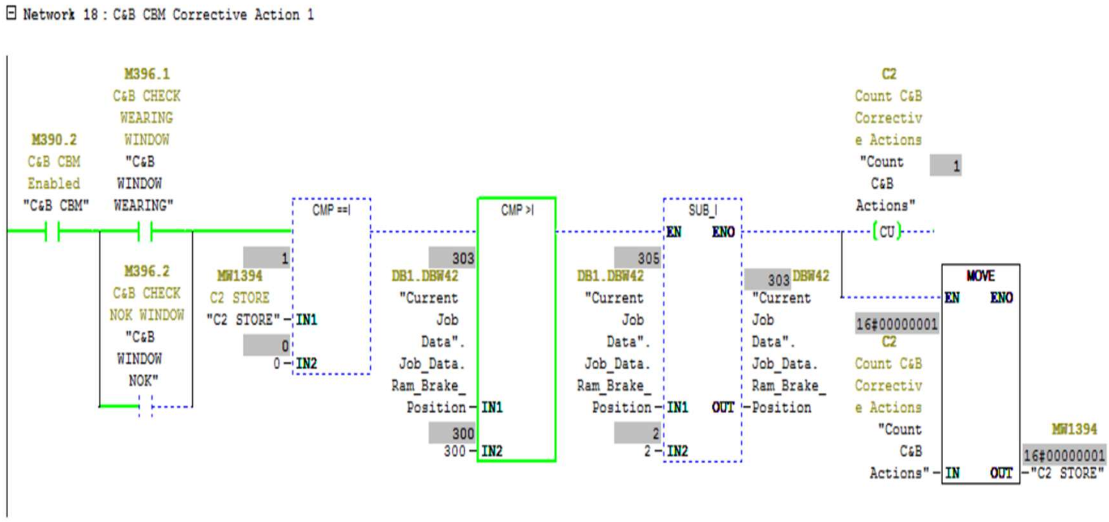


Figure 5.16: PLC Program NW18 Actioned Online

The second corrective action began in Network 19. The wearing condition and previous action completion started the 15-minute timer as showed in Figure 5.17.

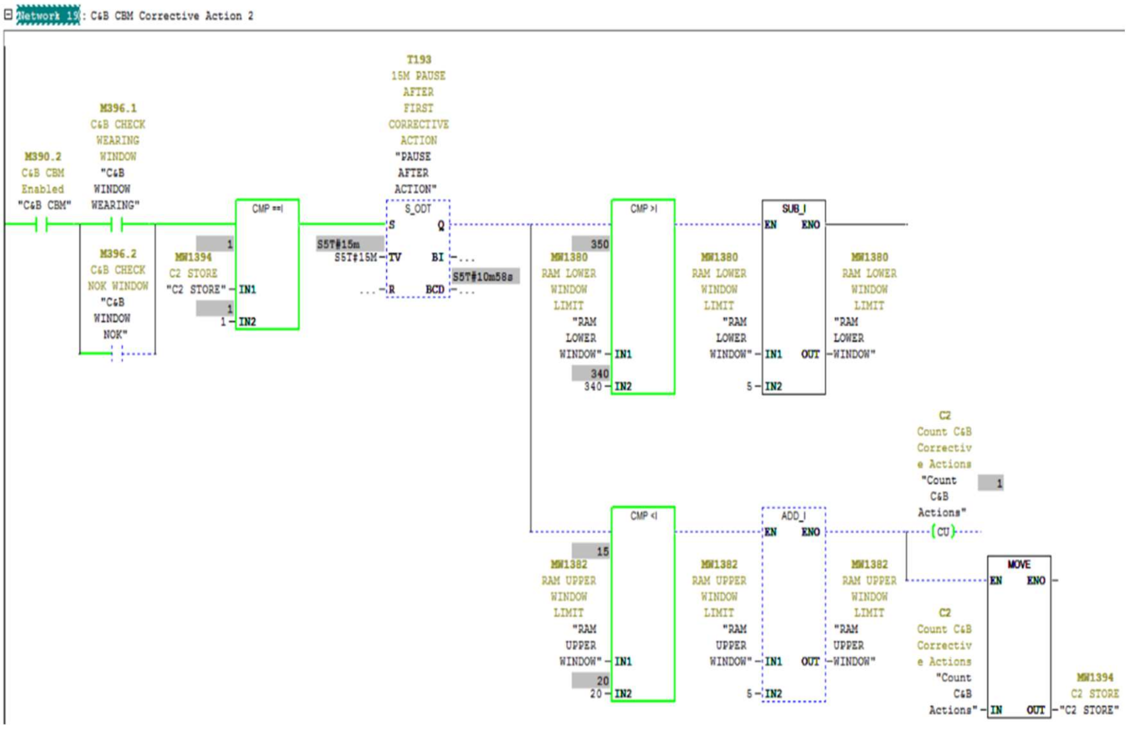


Figure 5.17: PLC Program NW19 Online

After 15 minutes passed the lower window limit went from 350° to 345° and the upper window limit changed from 15° to 20° (Figure 5.18) and ‘C2 Store’ increased to 2.

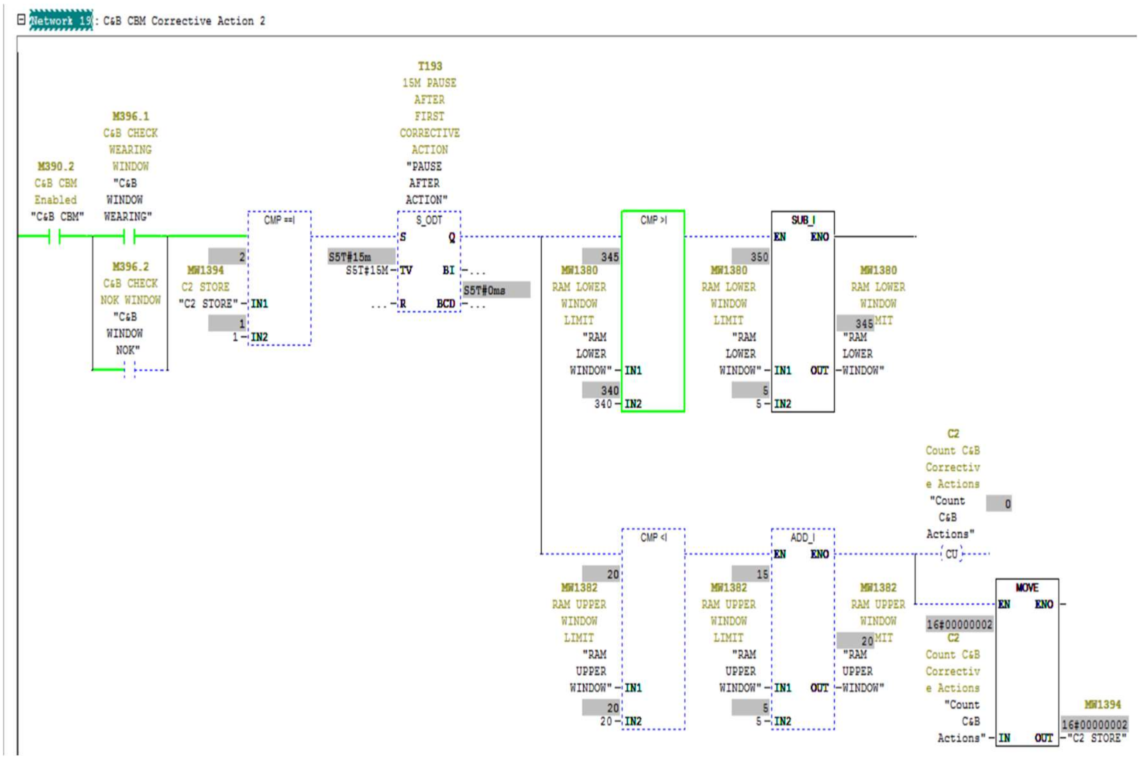


Figure 5.18: PLC Program NW19 Actioned Online

Once the second action finished, network 20 reset the ‘C2’ counter and updated the ‘C2 Store’ value to 0 as showed in Figure 5.19.

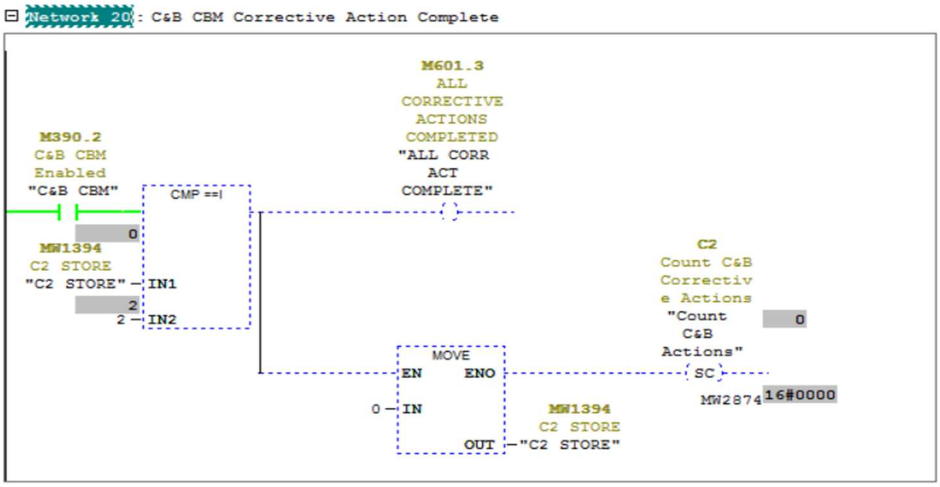


Figure 5.19: PLC Program NW20 Online

The VA sensor had minimal PLC integration. Network 22 (Figure 5.20) showed the activation of the 'Press at bottom' signal when the ram position was in the correct window.

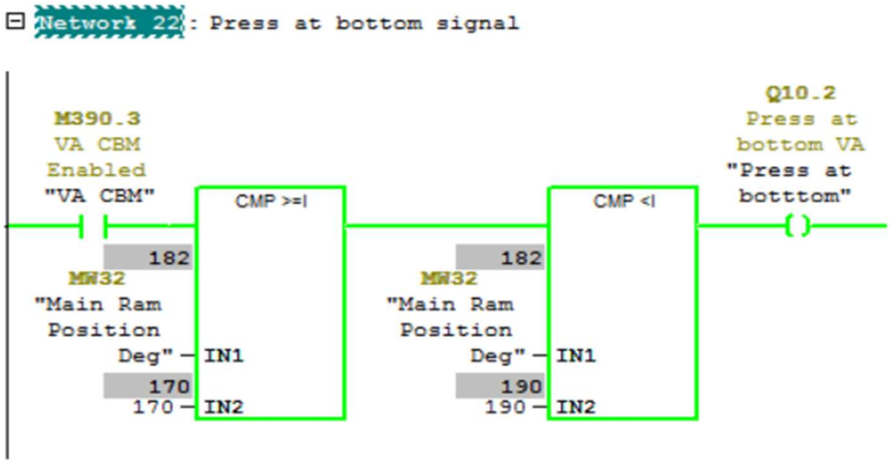


Figure 5.20: PLC Program NW22 Online

In network 23 the VA conditions were mapped from the PLC input to the PLC memory bit and network 24 mapped these bits to the alarm memory bits.

Network 26-35 mapped the bracket inputs to the corresponding memory bits while also updating the 'Press NIPP' memory word. Network 36-38 mapped the NIPP I/O and network 39 mapped the bracket signals to the alarm bits. Network 40-43 were simulated to test the pump shutoff in case of leak functionality – because there was no leak during testing on the pipe. Figure 5.21 showed that the preconditions were met. The most recently stored reading – meaning 'n-1' – was 100bar and the 'Upper bound of current pressure' – 'n' was 10bar. Therefore, the rate of change was calculated as '10' which was greater than '5'; this resulted in 'C3 Store' incrementing to 1.

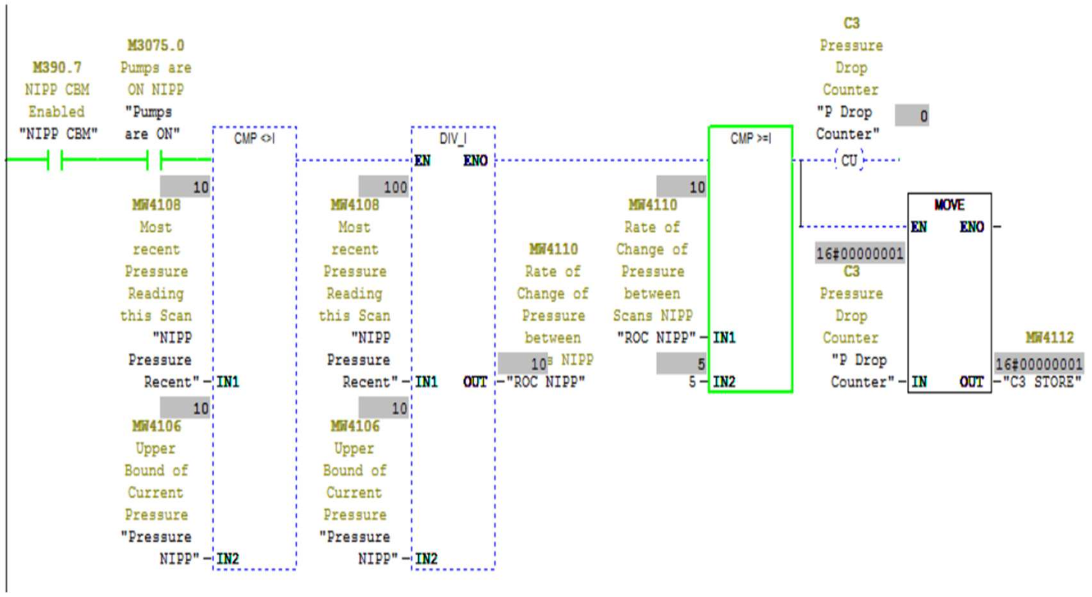


Figure 5.21: PLC Programming NW40 Online

Network 41 (Figure 5.22) showed the continuation of the detected leak programming.

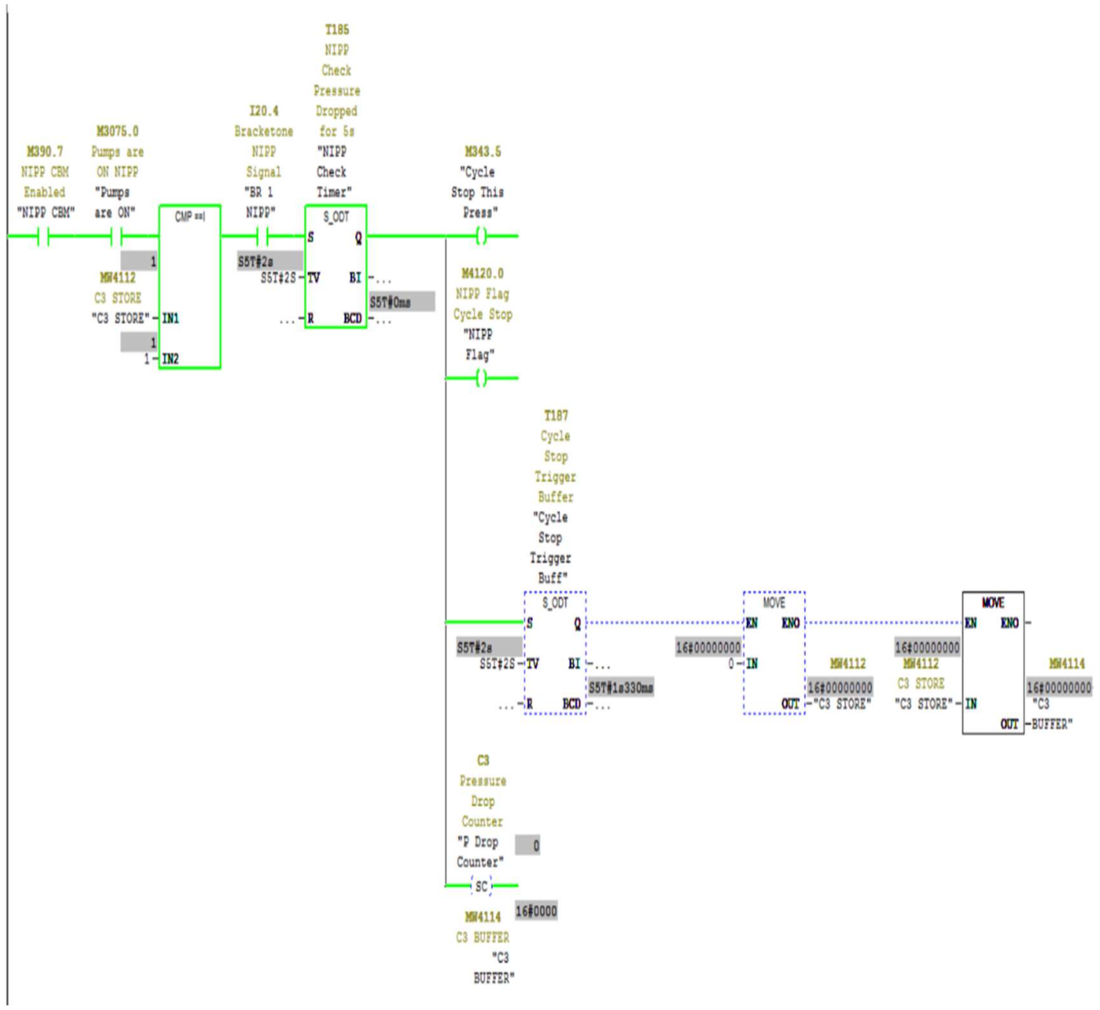


Figure 5.22: PLC Programming NW41 Online

The preconditions were met and the pressure remained at the lowest bracket for 2 seconds – waiting for 2 seconds removed any potential impact of interference misreporting 0bar. The press was then told to cycle stop and was given 2 seconds to do so. The counter 'C3' was then reset after 2 seconds, however during this period network 42 orchestrated the pumps shutting off, the preconditions necessitated that the press was cycle stopped and the pumps were still on initially. A slight pause of 500ms ensured the press was completely still and then the pumps were shutoff via a signal command as showed in Figure 5.23.

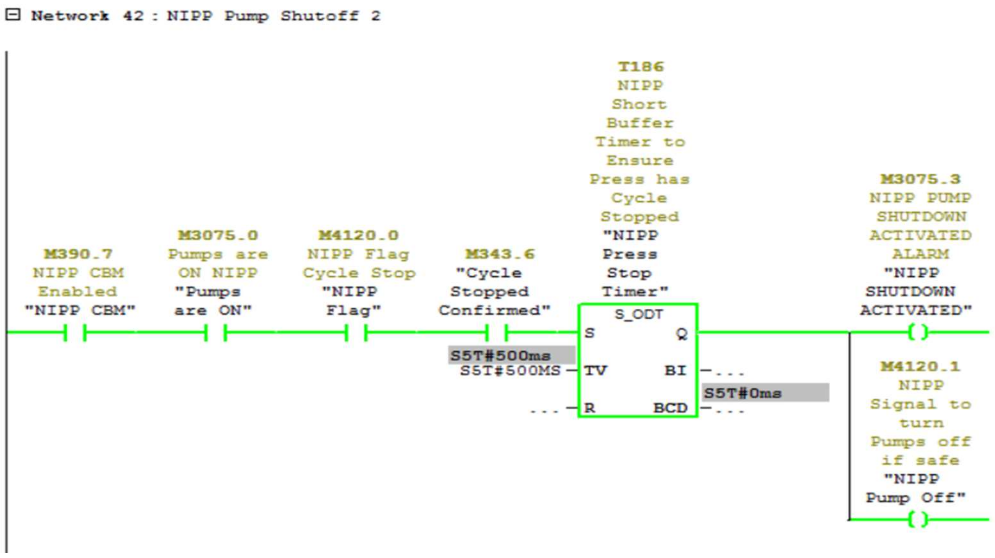


Figure 5.23: PLC Programming NW42 Online

After this, C3 was reset and the press had stopped. Network 43 was placed at the end of the program to ensure the 'Pressure NIPP' memory word updated at the correct time, Figure 5.24.

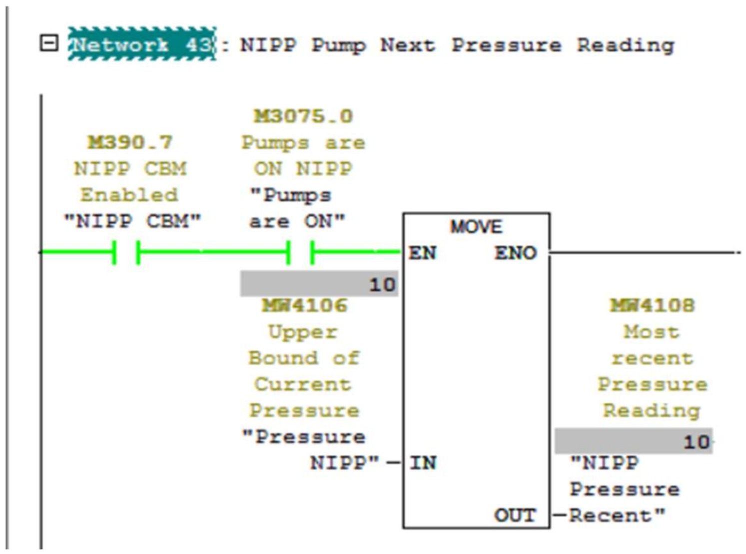


Figure 5.24: PLC Programming NW43 Online

Overall, the program worked as planned and it proved important in helping improve CM practices.

5.7 Wireless Transmission and Central Graphical Device

The nRF network was tested inside the factory. Other devices transmitted wirelessly but there was no apparent interference because a different base frequency was used. The nRF transceivers were housed inside an enclosure, which reduced interference but also slightly reduced signal strength. However, there were no signal losses throughout the entire test process: the network operated from the presses to the main office - a distance of ≈15m. The transmission system of characters ensured that there were no misinterpretations of condition. The portable powerpack was capable of supplying the entire CGD system with no current or voltage drops. An external view of the CGD in Figure 5.25 showed the nRF module connected to the Arduino which was sat on the rear of the screen which had a Raspberry Pi within.

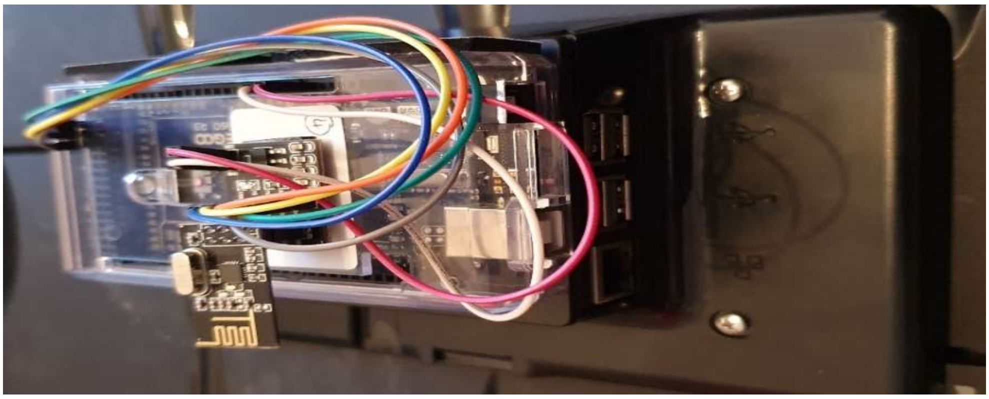


Figure 5.25: CGD External View

The GUI was populated numerous times with various CM statuses throughout the entire testing process. As showed by Figure 5.26 the C&B transmitted '1OK', VA sent '2WE' while the NIPP sent '3ER'.

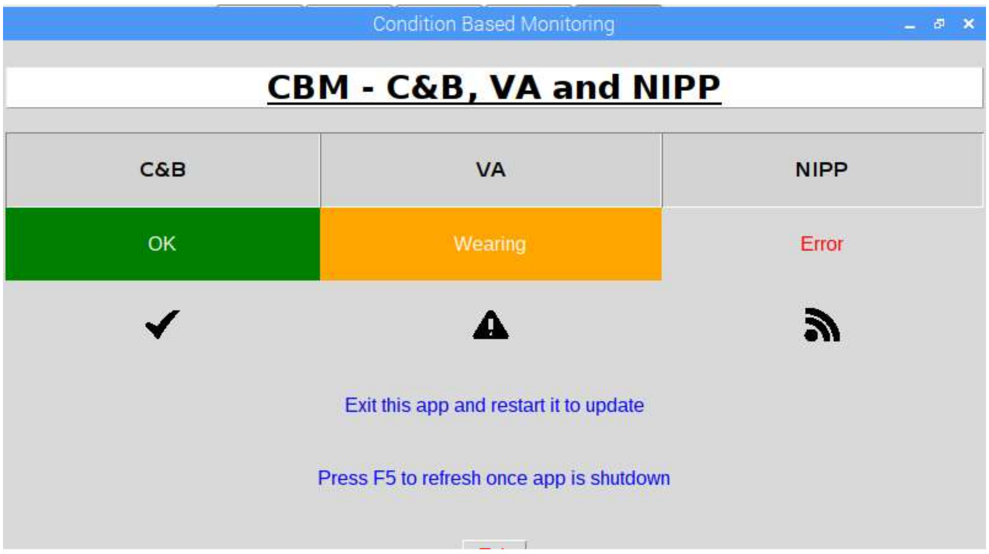


Figure 5.26: CGD GUI '1OK2WE3ER'

A further example of successful GUI generation was showed in Figure 5.27.

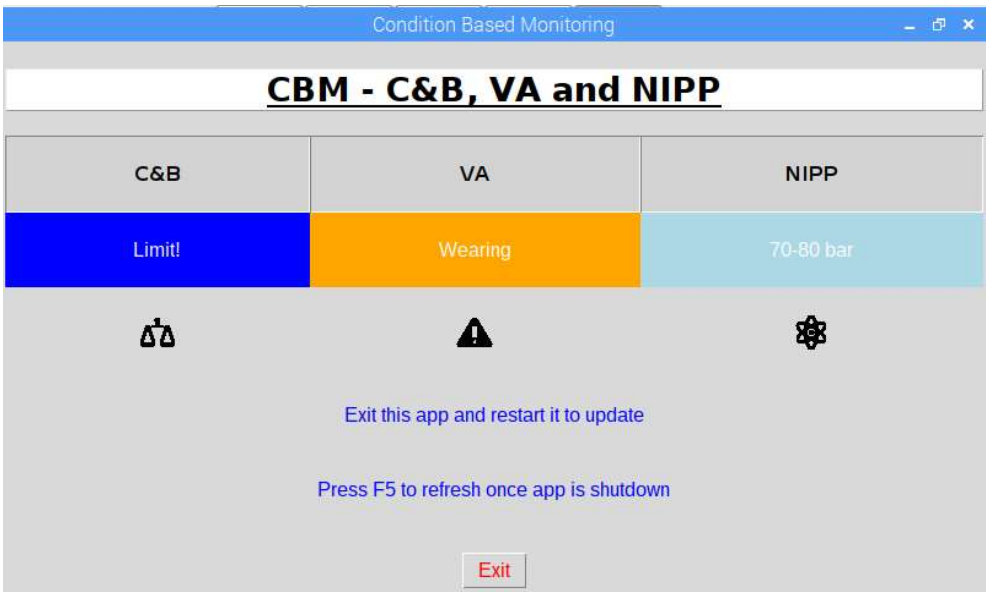


Figure 5.27: CGD GUI '1LE2WE308'

The C&B sensor was reading 'Limit Exceeded' as it was tested past its limits while the VA was wearing and the NIPP sensor spiked at 70-80bar during one of the early tests. The interference combatting techniques incorporated into the GUI program prevented incorrect data from being displayed. The Python Shell was a useful live feed of transmission even without the GUI.

5.8 Summary

Thorough tests in the factory on the equipment delivered insightful results that proved the functionality of the sensors individually and as a network. The results differed from their counterparts in Chapter 4 because of the different conditions encountered. Interfacing with the PLC proved successful and methods which were incorporated throughout the designs to reduce the effect of interference in CM activities were seen to be beneficial.

6 Conclusion and Future Work

6.1 Conclusion 6-101
6.2 Future Work 6-103

6.1 Conclusion

As stated initially, the research question was: Is it possible to design and implement a system for machine monitoring, establish possible interferences in the system and identify methods to minimise them.

Firstly, it was possible to design and implement a system for machine monitoring; 3 sensors were designed for specific machine monitoring purposes and they generated accurate data and condition status information. Secondly, several features were utilised to minimise possible interferences in the system; these ranged from physical electrical methods to digital programming methods. The sensors were designed to work automatically with minimum human intervention to provide the most useful CM system possible.

The C&B sensor produced respectable results in its final tests from Chapter 5. The first test returned an overall measurement range of 0.04cm which was 4% of the directly monitored range – this showed the sensor's capability. The correct condition was reported 80% of the time with the remaining 20% not reporting an incorrect condition but not reporting anything because the data it measured was insufficient for a conclusion to be drawn: further proving the beneficial effects of the programming methods that prevented incorrect reports being generated.

The second C&B test had a smaller measurement range of 0.03cm, only 3% of the directly monitored 1cm range. Compared to the theoretical requirements of being able to differentiate 8% of the 1cm range from other such ranges; achieving 3-4% is twice as good as required and a success – it is a vast improvement on the prior manual dipstick method. The reporting accuracy was 100% in the second test which gave an average of 90% correct actual condition reporting.

The VA sensor for the first final test was required to report a fundamental frequency of 24.8Hz but reported 25Hz instead which was 0.02Hz different – a mere 0.08% deviation. The 4th, 5th and 11th harmonic were detected as 98Hz, 123Hz and 270Hz when in theory they were 100Hz, 125Hz and 275Hz - a difference of 2%, 1.6% and 1.82% respectively. These small deviations were not obstructive to correctly reporting the condition of the motor; when the motor was mechanically loose it was reported as such.

The second VA test measured the peak frequency at 0.026g which was well within the required amplitude range. The non-peak vibration readings should have ideally been 0g, but as was showed in the 6000RPM test they were measured as being less than 0.0025g; approximately 10% of the peak value. This large difference between peaks and non-peaks ensured the program did not detect the smaller amplitudes as peaks; furthermore, the effectiveness of the interference reduction methods was clear.

The NIPP sensor generated results from 2 tests. The first test was completed at 0bar – where the strain theoretically should have been 0 hence no pressure reading. The range of readings was 1.17bar with the average across all 30 readings being 1.67bar – this is higher than it should have been and it showed that the values generated from the theoretical calculations were not completely accurate due to the assumptions which had to be made. Nevertheless, due to the other interference reduction methods developed – namely the removal of this 0bar offset from subsequent readings – it was ensured that the readings were still correctly within the 10bar window.

The second NIPP test monitored more readings during press cycles and noted the rate of change. There was no leak so the rate of change should not have exceeded 5 – it peaked at 3. The wide ranges within each pressure bracket recording provided emphasis on the need for data averaging and for initially setting 10bar as the bracket size. The smallest range was 1.57bar and the largest was 8.12bar; this suggested reduction of the window was not particularly feasible with the current setup although the median of these ranges was 3.51bar which was more reassuring.

All sensors were interfaced with the PLC and its I/O which enabled enhanced control. The modular programming and circuit designs also helped with fault finding and ensured that all sensors had the same protections in place. The wireless transmission of data – with programming methods and standards to reduce the effect of erroneous transmissions – boosted the usefulness of the system in the work environment.

The circuits were protected from interferences which would have affected the CM results in numerous ways. Capacitors were frequently used to ensure stable voltages were delivered to circuits, soldered solid connections were used to prevent miscellaneous signal loss and all circuit boards were enclosed in IP rated enclosures. The external cabling – where possible – was shielded, insulated and physically isolated from high power lines and existing network communication cables. Accessible control switches and MCBs were used to protect the main control system from electrical faults. Other methodologies used to minimise the effects of interferences in the system involved having the sensors measure at the opportune period which relied upon successful communication between the sensors and the PLC. The PLC sent signals which the sensors used to know when to take measurements, for example the VA sensor did not measure when the press was at bottom dead centre because of the extra irrelevant vibrations it would have recorded. Furthermore, methods were employed in the programming to further reduce possible errors that were identified. The averaging of values and marking certain values as ‘Limit Exceeded’ helped to remove the impact large erroneous errors may have caused. Employing a rigid and formal messaging structure over the nRF wireless transmission network ensured that any messages which differed from the structure were not recognised as valid.

In conclusion, the combination of these interference reduction methods and sensor designs for machine monitoring enabled the main aim and objectives to be achieved and a definite original contribution was demonstrated.

6.2 Future Work

The sensors were designed in a modular repeatable manner. Therefore, in the future further CM sensors could be designed and added to this multi-sensor system relatively quickly. The Raspberry Pi 3B+ models used in this research came with built in WiFi; a future development would be to incorporate the system into a WiFi network - if one was created - so that remote viewing and analysis of data would be easier – creating an IoT system.

The C&B sensor could have an adjustable offset permanently included meaning the system could be deployed on pre-worn pads and the NIPP sensor could hold variables for the pipe parameters and more equations if the processing time was not affected negatively. The VA sensor CM aspect could be modified further so that checks could take place to ensure the frequency peaks detected were harmonics before processing the data – although this was omitted initially so that noisy data was noticed. Also, more conditions being checked could enhance the breadth of use for the VA sensor. Other detrending and drift removal techniques could have been explored where the linear detrend was not the best method to use. A deliberated future modification to the NIPP sensor was to improve the accuracy of its measurements by reducing the averaging windows and alternatively using a separate higher resolution ADC for more precise pressure readings. The physical sizes of the enclosures and circuitry could be reduced and printed circuit boards could be mass manufactured for large scale deployment across the factory.

7 References

The reference entries are in alphabetical order.

A

Albarbar, A., *et al.* (2008) 'Suitability of MEMS accelerometers for condition monitoring: An experimental study', *Sensors,* 8(2), pp. 784-799.

Analog Devices (2015) *Data Sheet ADXL345.* Available at: https://www.analog.com/media/en/technical-documentation/data-sheets/ADXL345.pdf (Downloaded: 12 August 2019).

appJar (2018) *appJar.* Available at: http://appjar.info/ (Accessed: 30 March 2019).

Arduino (no date) *Arduino MEGA 2560 R3: Power*. Available at: https://store.arduino.cc/mega-2560-r3 (Accessed: 30 March 2019).

Arras, K. and Tipaldi, D. G. (2012) *Robotics 2 Target Tracking*. Available at: http://ais.informatik.uni-freiburg.de/teaching/ws11/robotics2/pdfs/rob2-19-tracking.pdf (Downloaded: 15 April 2019).

B

Benetti, M., *et al.* (2008) 'Pressure sensor based on surface acoustic wave resonators'. *Sensors, 2008 IEEE*, Lecce, Italy, 26-29 October. IEEE, pp. 1024-1027.

Berkeley (no date) *Capturing Images.* Available at: http://microscopy.berkeley.edu/courses/dib/sections/02Images/sampling.html (Accessed: 25 July 2019).

Bluetooth (2019) *Radio* Versions. Available at: https://www.bluetooth.com/bluetooth-technology/radio-versions (Accessed: 30 March 2019).

Bolton, W. (2015) 'Input/Output Devices', in Bolton, W. *Programmable logic controllers.* 6th edn. London: Newnes.

Bolutek (no date) *BLK-MD-BC04-B Bluetooth Module.* Available at: http://kazus.ru/forums/attachment.php?attachmentid=66390&d=1403886443 (Downloaded: 30 March 2019).

Boston University Physics (no date) *Temperature Dependence of Resistance.* Available at: http://physics.bu.edu/~duffy/sc526_notes05/Rtemperature.html (Accessed: 30 March 2019).

Brüel & Kjær (1982) *Measuring Vibration*. Available at: https://www.bksv.com/media/doc/br0094.pdf (Downloaded: 30 March 2019).

Burr-Brown (2009) *Instrumentation Amplifier with Precision Voltage Reference.* Available at: http://www.ti.com/lit/ds/sbos060/sbos060.pdf (Downloaded: 07 October 2019).

C

Carullo, A. and Parvis, M. (2001) 'An ultrasonic sensor for distance measurement in automotive applications', *IEEE Sensors Journal,* 1(2), pp. 143-147.

Claycomb, T. (2015) *Single-Supply Strain Gauge in a Bridge Configuration Reference Design.* Available at: http://www.ti.com/lit/ug/tidub00/tidub00.pdf (Accessed: 10 March 2019).

Components101 (2018) *nRF24L01 Wireless RF Module.* Available at: https://components101.com/wireless/nrf24l01-pinout-features-datasheet (Accessed: 04 December 2019).

Cytron (no date) *Breakout for NRF24L01 with Socket.* Available at: https://www.cytron.io/p-breakout-for-nrf24l01-with-socket (Accessed: 05 November 2019).

D

De Silva, C. W. (2015) *Sensors and actuators: Engineering system instrumentation.* 2nd edn. London: CRC Press.

E

Endolith (2011) *Peak Detect.* Available at: https://gist.github.com/endolith/250860 (Accessed: 05 November 2019).

Espressif Systems (2018) *ESP8266EX Datasheet.* Available at: https://www.espressif.com/sites/default/files/documentation/0a-esp8266ex_datasheet_en.pdf (Downloaded: 30 March 2019).

F

Fairchild Semiconductor (2011) *FOD814 Series, FOD817 Series 4-Pin High Operating Temperature Phototransistor Optocouplers.* Available at: https://docs.rs-online.com/82cd/0900766b81384330.pdf (Downloaded: 30 April 2019).

G

Galipeau, D. W.*, et al.* (1997) 'Surface acoustic wave microsensors and applications', *Smart materials and structures,*6(6), p. 658.

Gandy, D. (2007) *Carbon Steel Handbook.* Available at: https://wiki.olisystems.com/wiki/images/0/0c/Carbon-Steel-Handbook.pdf (Downloaded: 14 May 2019).

Gilchrist, A. (2016) *Industry 4.0: the industrial internet of things*. Apress.

Guadalupi, A. (2019) *Arduino MEGA 2560 R3 Schematic Diagram*. Available at: https://content.arduino.cc/assets/MEGA2560_Rev3e_sch.pdf (Downloaded: 30 March 2019).

H

Hansford Sensors (2020) *The Hansford Vibration Monitoring App.* Available at: https://www.hansfordsensors.com/resources/vibration-calculator/ (Accessed: 17 March 2020).

HBM (no date) *Measuring with strain gauges: How to prevent unwanted temperature effects on your measurement result.* Available at: https://www.hbm.com/en/6725/article-temperature-compensation-of-strain-gauges/ (Accessed: 15 April 2019).

HobbyComponents (no date) *BF350-3AA Strain Gauge Sensor*. Available at: https://hobbycomponents.com/sensors/962-bf350-3aa-strain-gauge-sensor (Accessed: 24 April 2019).

Hobbytronics (no date) *Teensy v3.6*. Available at: http://www.hobbytronics.co.uk/teensy-v36 (Accessed: 30 March 2019).

Horowitz, P. and Hill, W. (2015) *The art of electronics.* 3rd edn. Cambridge: Cambridge Univ. Press. (Accessed: 14 March 2019).

I

ITead Studio (2010) *HC-05 Bluetooth to Serial Port Module*. Available at: http://www.electronicaestudio.com/docs/istd016A.pdf (Downloaded: 30 March 2019).

J

Jones, C. (2008) *Non-contact displacement sensor technologies.* Available at: https://www.micro-epsilon.co.uk/press/publication/pub-uk--2008-10--power-in-motion.pdf (Downloaded: 05 March 2019).

K

Kadapa, C. (2017) *Advanced Structural Analysis – Cylinders under Pressure.* Available at: http://engweb.swan.ac.uk/~c.kadapa/teaching/2017-2018/EGF316/week2/EGF316%20Thin%20and%20Thick%20Cylinders%20-%20notes.pdf (Downloaded: 31 March 2019).

L

Leens, F. (2009) 'An introduction to I2C and SPI protocols', *IEEE Instrumentation & Measurement Magazine,* 12(1), pp. 8-13.

Looney, M. (2014) *An Introduction to MEMS Vibration Monitoring.* Available at: http://www.analog.com/en/analog-dialogue/articles/intro-to-mems-vibration-monitoring.html (Accessed: 17 April 2019).

Lowrie, W. (2007) *Fundamentals of geophysics.* 2nd edn. Cambridge: Cambridge Univ. Press.

Lyons, R.G. (2004) 'The Fast Fourier Transform', in Lyons, R.G. *Understanding Digital Signal Processing.* 2nd edn. New Jersey: Pearson Education.

M

Mais, J. (2002) *Spectrum Analysis: The key features of analysing spectra*. Available at: https://www.skf.com/binary/tcm:12-113997/CM5118%20EN%20Spectrum%20Analysis.pdf (Downloaded: 18 March 2020).

MatPlotLib (2020) *Matplotlib: Visualization with Python.* Available at: https://matplotlib.org/ (Accessed: 06 February 2020).

Mobley, R. K. (2002) *An Introduction To Predictive Maintenance.* 2nd edn. London: Butterworth Heinemann.

Mohammad, T. (2009) 'Using ultrasonic and infrared sensors for distance measurement', *World Academy of Science, Engineering and Technology,* 51, pp. 293-299.

N

Nagimov, R. (2016) *ADXL345 SPI Diagram.* Available at: www.github.com/nagimov/adxl345spi (Accessed: 06 February 2020).

National Instruments (1998) *Strain Gauge Measurement – A Tutorial.* Available at: http://elektron.pol.lublin.pl/elekp/ap_notes/NI_AN078_Strain_Gauge_Meas.pdf (Downloaded: 28 March 2019).

Neale, M.J. and Woodley, B.J. (1975) *Condition Monitoring Methods and Economics.* Available at: https://www.bksv.com/media/doc/16-054.pdf (Downloaded: 23 March 2019).

Nordic Semiconductor (2006) *Single chip 2.4 GHz Transceiver nRF24L01.* Available at: https://www.sparkfun.com/datasheets/Components/nRF24L01_prelim_prod_spec_1_2.pdf (Downloaded: 23 March 2019).

Nordic Semiconductor (2008) *nRF24L01+ Single Chip 2.4GHz Transceiver.* Available at: https://components101.com/sites/default/files/component_datasheet/nRF24L01%20Datasheet.pdf (Accessed: 04 November 2019).

NumPy (2019) *NumPy Documentation.* Available at: https://numpy.org/doc/ (Accessed: 28 December 2019).

O

Orhan, S., Aktürk, N. and Celik, V. (2006) 'Vibration monitoring for defect diagnosis of rolling element bearings as a predictive maintenance tool: Comprehensive case studies', *Ndt & E International,* 39(4), pp. 293-298. doi: 10.1016/j.ndteint.2005.08.008

P

Pandas (2020) *Pandas Documentation.* Available at: https://pandas.pydata.org/docs/ (Accessed: 08 February 2020).

Penman, J. and Tavner, T. (1989) *Condition monitoring of electrical machines*. New York: Research Studies Press Ltd.

PJRC (no date) *Welcome to Teensy 3.6*. Available at: https://www.pjrc.com/teensy/card9a_rev1.pdf (Downloaded: 30 March 2019).

R

Raspberry Pi (no date) *Raspberry Pi 3 Model B+*. Available at: https://static.raspberrypi.org/files/product-briefs/Raspberry-Pi-Model-Bplus-Product-Brief.pdf (Downloaded: 30 January 2019).

Recom (2014) *DC/DC-Converter 500mA Single Output.* Available at: https://docs.rs-online.com/dc35/0900766b81373235.pdf (Downloaded: 13 May 2019).

S

Sadeghioon, A., *et al.* (2014) 'Design and development of a nonintrusive pressure measurement system for pipeline monitoring', *Journal of Pipeline Systems Engineering and Practice,* 5(3), doi: 10.1061/(ASCE)PS.1949-1204.0000169

SciPy (2019) *SciPy Documentation.* Available at: https://docs.scipy.org/doc/scipy/reference/index.html (Accessed: 25 January 2020).

SciPy Cookbook (2015) *SciPy Cookbook*. Available at: https://scipy-cookbook.readthedocs.io/index.html (Accessed: 25 January 2020).

SciPy Lectures (no date) *Detrending a signal.* Available at: https://scipy-lectures.org/intro/scipy/auto_examples/plot_detrend.html (Accessed: 25 January 2020).

Sharp (no date) *GP2Y0A41SK0F*. Available at: http://www.farnell.com/datasheets/2364614.pdf (Downloaded: 30 March 2019).

Siemens (2019) *6ES7321-1BL00-0AA0.* Available at:https://support.industry.siemens.com/cs/pd/390860?pdti=td&dl=en&lc=en-WW. (Accessed: 30 March 2019).

Siemens Simcenter (2019) *Root Mean Square (RMS) and Overall Level.* Available: https://community.sw.siemens.com/s/article/root-mean-square-rms-and-overall-level. (Accessed: 15 February 2020).

Simmons, S. (2020) *UnoArduSim*. Available at: https://www.sites.google.com/site/unoardusim/services (Accessed: 20 Jan 2020).

Sinha, S., Gopal, R. and Mukhiya, R. (2014) 'Design and simulation of MEMS differential capacitive accelerometer'. *Proceeding of ISSS international conference on smart materials, structures and systems*, doi: 10.13140/2.1.1074.8809

SKF (2000) *Vibration Diagnostic Guide.* Available at: http://edge.rit.edu/edge/P14453/public/Research/SKF%20VibrationGuide.pdf (Downloaded: 03 February 2020).

Smith, W. S. (2011) *The Fast Fourier Transform.* Available at: https://www.dspguide.com/ch12/2.htm (Accessed: 16 March 2019).

ST (2013) *LD1117 – Adjustable and fixed low drop positive voltage regulator*. Available at: https://www.st.com/content/ccc/resource/technical/document/datasheet/99/3b/7d/91/91/51/4b/be/CD00000544.pdf/files/CD00000544.pdf/jcr:content/translations/en.CD00000544.pdf (Downloaded: 22 June 2019).

T

Tandon, N. and Choudhury, A. (1999) 'A review of vibration and acoustic measurement methods for the detection of defects in rolling element bearings', *Tribology international,* 32(8), pp. 469-480. Available at: http://eprint.iitd.ac.in/handle/2074/695.

Tavner, P.*, et al.* (2008) *Condition monitoring of rotating electrical machines.* IET.

TE Connectivity (2017) *Choosing the right type of accelerometer.* Available at: https://www.mouser.com/pdfdocs/choosing-the-right-accelerometer-white-paper.pdf (Downloaded: 11 July 2019).

ThePiHut (no date) *Raspberry Pi 3 Model B Plus*. Available at: https://thepihut.com/products/raspberry-pi-3-model-b-plus (Accessed: 30 March 2019).

TMRh20 (2020) *Optimized high speed Nrf24l01+ driver class documentation.* Available at: http://tmrh20.github.io/RF24/ (Accessed: 04 January 2020).

U

University of Cambridge (no date) *Semiconductor Physics Group – Surface Acoustic Waves.* Available at: https://www.sp.phy.cam.ac.uk/research/fundamentals-of-low-dimensional-semiconductor-systems/saw (Accessed: 30 April 2019).

V

Varanis, M.*, et al.* (2018) 'MEMS accelerometers for mechanical vibrations analysis: a comprehensive review with applications', *Journal of the Brazilian Society of Mechanical Sciences and Engineering,* 40(11), p. 527.

W

WatElectronics (2019) *ASCII Numbering System and its Conversions from ASCII to Hex.* Available at: https://www.watelectronics.com/ascii-numbering-system-and-its-conversion-from-ascii-to-hex/ (Accessed: 15 November 2019).

Whitlock, B. (2008) *Understanding, Finding & Eliminating Ground Loops*. Available at: http://web.mit.edu/jhawk/tmp/p/EST016_Ground_Loops_handout.pdf (Downloaded: 11 May 2019).

X

XP Power (2017) *JTD20 Series DC-DC Converter.* Available at: https://docs.rs-online.com/8037/0900766b815a94f9.pdf (Accessed: 10 March 2019).

8 Appendices

8.1 A1 Clutch and Brake Arduino Program excluding nRF 8-110
8.2 A2 Clutch and Brake Interfacing and Calibration Circuit Schematic 8-115
8.3 A3 Vibration Analysis ADXL345 SPI C Program (Nagimov, 2016) 8-116
8.4 A4 Vibration Analysis ADXL345 FFT Program 8-121
8.5 A5 Vibration Analysis Interfacing Circuit Schematic 8-129
8.6 A6 Non-Intrusive Pipe Pressure Arduino Program excluding nRF 8-130
8.7 A7 Non-Intrusive Pipe Pressure Interfacing and Calibration Circuit 8-140
8.8 A8 Non-Intrusive Pipe Pressure Measuring Circuit Schematic 8-142
8.9 A9 9V Regulator Circuit Schematic 8-143
8.10 A10 Clutch and Brake Pad Wear Test Transmission Program 8-144
8.11 A11 Non-Intrusive Pipe Pressure Test Transmission Program 8-145
8.12 A12 Vibration Analysis Serial Test Transmission Program 8-146
8.13 A13 Vibration Analysis Arduino Test Transmission Program 8-147
8.14 A14 Central Graphical Device Receiver Test Transmission Program 8-148
8.15 A15 Central Graphical Device GUI Test Transmission Program 8-149
8.16 A16 Siemens PLC Integration STL Program 8-152
8.17 A17 Clutch and Brake Sensor Prototype Test Result Graphs 8-161
8.18 A18 Connections Junction Box Schematics 8-167
8.19 A19 C&B Interfacing Circuit Photograph and Terminal Designations 8-171
8.20 A20 Complete C&B Arduino Program 8-172
8.21 A21 C&B Implementation Test Results 1 8-178
8.22 A22 C&B Implementation Test Results 2 8-179
8.23 A23 VA Interfacing Circuit Photograph and Terminal Designations 8-180
8.24 A24 NIPP Measuring Circuit Photograph and Terminal Designations 8-181
8.25 A25 NIPP Interfacing Circuit Photograph and Terminal Designations 8-182
8.26 A26 Complete NIPP Arduino Program 8-183
8.27 A27 NIPP Pressure Readings Graph 1 8-194
8.28 A28 NIPP Pressure Readings Graph 2 8-195
8.29 A29 9V Regulator Circuit Photograph and Terminal Designations 8-196

8.1 A1 Clutch and Brake Arduino Program excluding nRF

```
/* Input and Output Declarations */

#define sensor A0                 // Analog input from the IR sensor

const int calib = 30;   // Calibration button input to set a datum - hold for 5 seconds
const int presstdc = 31;          // Press at TDC signal from the PLC
const int pressmotive = 33;       // Press Motive signal from the PLC
const int lifebitin = 34;         // Life Bit signal from the PLC
const int lifebitout = 35;        // Life Bit signal to the PLC
const int calibcomp = 36;         // Calibration complete to allow measurement
const int OK = 37;                // Current condition OK signal to the PLC
const int Wearing = 38;           // Current condition Wearing signal to the PLC
const int NOK = 39;               // Current condition NOK signal to the PLC
const int LimitExceeded = 40;
// The boundary limits have been exceeded signal to the PLC

/* Setup Local Variables */

float difference0 = 0.000;        // First hold area for difference measurement
float difference1 = 0.000;        // Second hold area for difference measurement
float difference2 = 0.000;        // Third hold area for difference measurement
float difference3 = 0.000;        // Fourth hold area for difference measurement
float difference4 = 0.000;        // Fifth hold area for difference measurement
float difference5 = 0.000;        // Sixth hold area for difference measurement
float difference6 = 0.000;        // Seventh hold area for difference measurement
float difference7 = 0.000;        // Eighth hold area for difference measurement
float difference8 = 0.000;        // Ninth hold area for difference measurement
float difference9 = 0.000;        // Tenth hold area for difference measurement
float checkreading;               // Check each reading before confirmation
float avgdifference;              // Average of all the averages
float calibdistance;              // The initial measured calibration distance
float avgone;
// The first average from the first ten measurements
float avgtwo;
// The second average from the second ten measurements
float avgthree;
// The third average from the third ten measurements

int calibcompinternal;
// Internal bit for calibration completion checking
int avgdifferencefirstcomp;       // Checks the first avg difference has been taken


void setup() {

  pinMode(calib, INPUT);                  // Set calib pin as an input
  pinMode(presstdc, INPUT);               // Set presstdc pin as an input
  pinMode(pressmotive, INPUT);            // Set pressmotive pin as an input
  pinMode(lifebitin, INPUT);              // Set lifebitin pin as an input
  pinMode(lifebitout, OUTPUT);            // Set lifebitout pin as an output
  pinMode(OK, OUTPUT);                    // Set OK pin as an output
  pinMode(Wearing, OUTPUT);               // Set Wearing pin as an output
  pinMode(NOK, OUTPUT);                   // Set NOK pin as an output
  pinMode(calibcomp, OUTPUT);             // Set calibcomp pin as an output
  pinMode(LimitExceeded, OUTPUT);         // Set LimitExceed as an output
  digitalWrite(calibcomp, LOW);           // Set calibcomp low on power cycle
        }

```

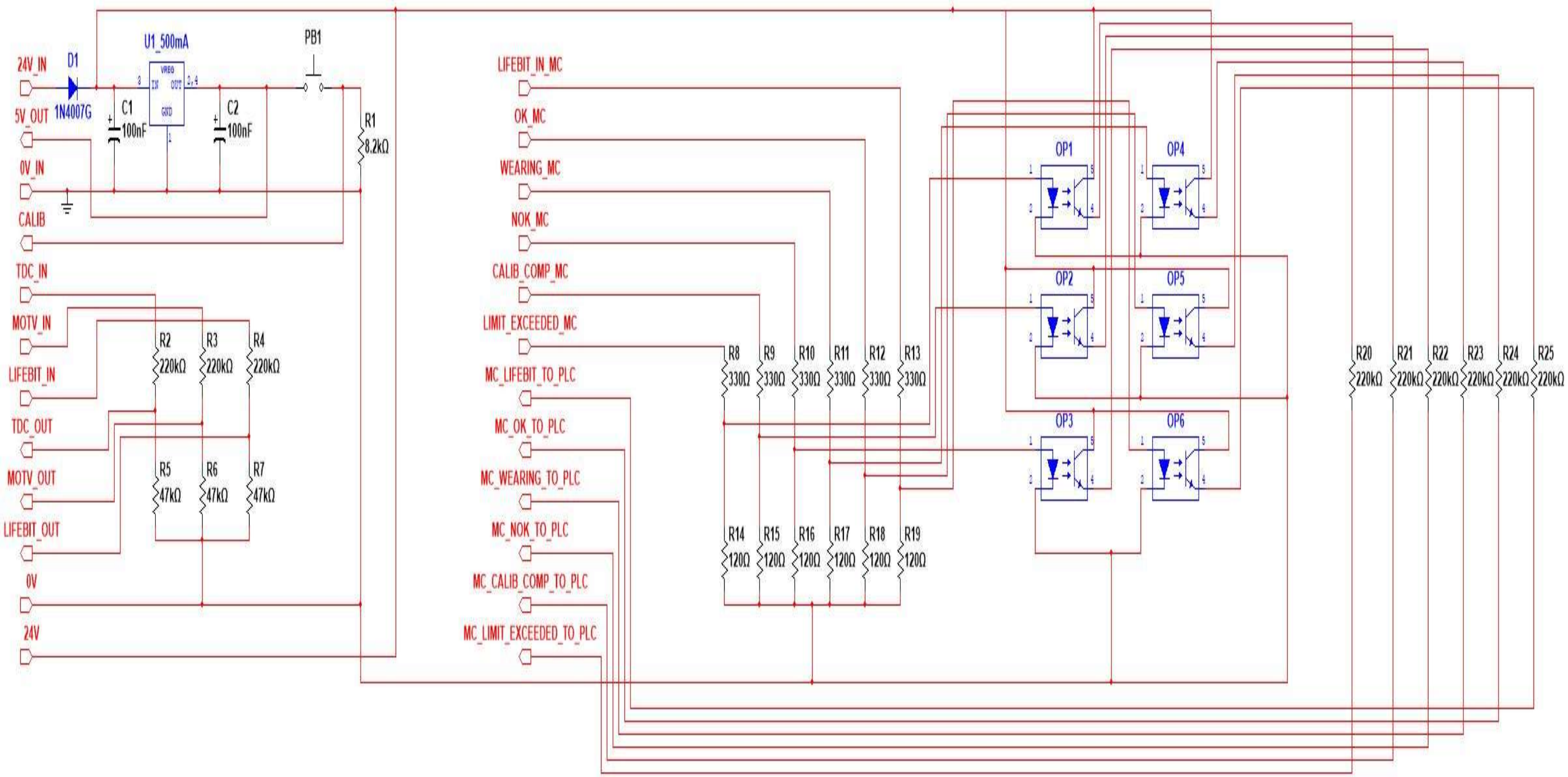
```
void loop() {
 if (digitalRead(lifebitin) == HIGH){
// Check lifebit from PLC is OK
   digitalWrite(lifebitout, HIGH);
// Send lifebit to PLC
   if (digitalRead(presstdc) == HIGH & digitalRead(pressmotive) == LOW){
// Check for Press at TDC and Press Motive off
     if (digitalRead(calib) == HIGH){
// Check calibration button is pressed
       float measurement = analogRead(sensor);
// Read the value from the sensor
       float measasvolt = (measurement * 0.0048828125);
// Convert value from sensor to a voltage
       calibdistance =  (((13.147)/(measasvolt))-0.42)) );
// Use the equation to convert to a distance
       delay(500);
// Pause 0.5s
       digitalWrite(calibcomp, HIGH);
// Set calibcomp signal high to PLC
       calibcompinternal = HIGH;
// Set internal calibration complete signal high
                                 }
       delay(100);
// Pause 0.1s
   if (calibcompinternal == HIGH){
// Only allow for measurement to take place if calibrated
    for (int j = 0; j<=2; j++){
// Initialise the For Loop for 3 sets of averages
     for (int i = 0; i <=9; i++) {
// Initialise the For Loop for 10 measurements
      chk:
// Pointer for a program jump
       float measurement = analogRead(sensor);
// Read the value from the sensor
          float measasvolt = measurement * 0.0048828125;
   // Convert value from sensor to a voltage
          float checkreading = ((calibdistance - (((13.147)/(measasvolt))-0.42)) );
   // Temporary storage for difference measurement
          delay(100);                                          // Pause 0.1s
            if (checkreading > 2){
   // Check the reading to see if it is over 2
             goto chk;
   // If this is the case then measure again
             delay(100);        }
   // Pause 0.1s
          if (i == 0)     {
   // The initial measurement
           difference0 = ((calibdistance - (((13.147)/(measasvolt))-0.42)));
   // Find the difference by subtracting measured distance from calibration distance
           delay(100);   }                                     // Pause 0.1s
```

```
        if (i == 1)     {
// The second measurement

            difference1 = ((calibdistance - (((13.147)/(measasvolt))-0.42)));
           // Find difference by subtracting measured distance from calibration distance
            delay(100);   }                              // Pause 0.1s

        if (i == 2)     {
// The third measurement

            difference2 = ((calibdistance - (((13.147)/(measasvolt))-0.42)));
// Find difference by subtracting measured distance from calibration distance
            delay(100);   }                              // Pause 0.1s

        if (i == 3)     {                  // The fourth measurement

            difference3 = ((calibdistance - (((13.147)/(measasvolt))-0.42)));
// Find difference by subtracting measured distance from calibration distance
            delay(100);   }
// Pause 0.1s

        if (i == 4)     {
// The fifth measurement

            difference4 = ((calibdistance - (((13.147)/(measasvolt))-0.42)));
// Find difference by subtracting measured distance from calibration distance
            delay(100);   }
// Pause 0.1s

        if (i == 5)     {
// The sixth measurement

            difference5 = ((calibdistance - (((13.147)/(measasvolt))-0.42)));
// Find difference by subtracting measured distance from calibration distance
            delay(100);   }
// Pause 0.1s

        if (i == 6)     {
 // The seventh measurement

            difference6 = ((calibdistance - (((13.147)/(measasvolt))-0.42)));
// Find difference by subtracting measured distance from calibration distance
            delay(100);   }
 // Pause 0.1s

        if (i == 7)     {
 // The eighth measurement

            difference7 = ((calibdistance - (((13.147)/(measasvolt))-0.42)));
// Find difference by subtracting measured distance from calibration distance
            delay(100);   }
 // Pause 0.1s

        if (i == 8)     {
 // The ninth measurement

            difference8 = ((calibdistance - (((13.147)/(measasvolt))-0.42)));
// Find difference by subtracting measured distance from calibration distance
            delay(100);   }
// Pause 0.1s
```

```
        if (i == 9)     {
 // The tenth measurement

          difference9 = ((calibdistance - (((13.147)/(measasvolt))-0.42)));
// Find difference by subtracting measured distance from calibration distance
          delay(100);    }
// Pause 0.1s
                            }
      delay(50);
// Pause 0.05s

      avgdifference = ((difference0 + difference1 + difference2 + difference3 + difference4
  + difference5 + difference6 + difference7 + difference8 + difference9)/(10));
// Average the differences after 10 measurements
      avgdifferencefirstcomp = HIGH;
// The first average has been completed

      delay(50);
// Pause 0.05s

        if (j == 0){
 // The first set of averages
        avgone = avgdifference; }
// The current average difference is copied to avgone hold
        delay(50);
// Pause 0.05s
        if (j == 1){
// The second set of averages
        avgtwo = avgdifference; }
// The current average difference is copied to avgtwo hold
        delay(50);
// Pause 0.05s
        if (j == 2){
// The third set of averages
        avgthree = avgdifference;}
// The current average difference is copied to avgthree hold
        delay(50);
// Pause 0.05s
                          }

     if(avgdifferencefirstcomp == HIGH){
// Check that a difference has been calculated before output

     if (avgone >= 0 && avgone <= 0.08 && avgtwo >= 0 && avgtwo <= 0.08 &&
avgthree >= 0 && avgthree <= 0.08){
// Check all average differences in a set range for OK

      digitalWrite(OK, HIGH);
// Measurement is OK - 1OK
      digitalWrite(Wearing, LOW);
// Unused output set to off
      digitalWrite(NOK, LOW);
 // Unused output set to off
      digitalWrite(LimitExceeded, LOW);
// The limits have not been exceeded

}

     else if (avgone > 0.08 && avgone <= 0.16 && avgtwo > 0.08 && avgtwo <= 0.16 &&
```

```
 avgthree > 0.08 && avgthree <= 0.16){
// Check all average differences in a set range for Wearing

     digitalWrite(OK, LOW);
// Unused output set to off
     digitalWrite(Wearing, HIGH);
// Measurement is Wearing - 1WE
     digitalWrite(NOK, LOW);
// Unused output set to off
     digitalWrite(LimitExceeded, LOW);
// The limits have not been exceeded
}

    else if (avgone > 0.16 && avgone <= 1.00 && avgtwo > 0.16 && avgtwo <= 1.00 &&
 avgthree > 0.16 && avgthree <= 1.00){
// Check all average differences in a set range for NOK

     digitalWrite(OK, LOW);
// Unused output set to off
     digitalWrite(Wearing, LOW);
// Unused output set to off
     digitalWrite(NOK, HIGH);
// Measurement is NOK - 1NO
     digitalWrite(LimitExceeded, LOW);
// The limits have not been exceeded
                                                                                                    }

    else if (avgone < 0 && avgtwo < 0 && avgthree < 0){
// Check if all average differences in a set are less than 0
     digitalWrite(OK, LOW);
// Unused output set to off
     digitalWrite(Wearing, LOW);
// Unused output set to off
     digitalWrite(NOK, LOW);
// Unused output set to off
     digitalWrite(LimitExceeded, LOW);
// The limits have been exceeded but it is classed as an error - 1ER
                                             }
    else if (avgone > 1.00 && avgtwo > 1.00 && avgthree > 1.00) {
// Check if all average differences in a set are more than 1 - should never occur
     digitalWrite(OK, LOW);
// Unused output set to off
     digitalWrite(Wearing, LOW);
// Unused output set to off
     digitalWrite(NOK, LOW);
// Unused output set to off
     digitalWrite(LimitExceeded, HIGH);
// The physical limits have been exceeded - 1LE
                                                            }
    delay(1000);
// Delay for transmission
    digitalWrite(lifebitout, LOW);
// Reset lifebit to PLC before next loop
   }
   }
   }
   }
   }
}
```

8.2 A2 Clutch and Brake Interfacing and Calibration Circuit Schematic



8.3 A3 Vibration Analysis ADXL345 SPI C Program (Nagimov, 2016)

```c
#include <stdio.h>
#include <pigpio.h>
#include <time.h>
#include <math.h>
#include <string.h>
#include <stdlib.h>

#define DATA_FORMAT   0x31  // data format register address
#define DATA_FORMAT_B 0x0B  // data format bytes: +/- 16g range, 13-bit resolution
(p. 26 of
 ADXL345 datasheet)
#define READ_BIT      0x80
#define MULTI_BIT     0x40
#define BW_RATE       0x2C
#define POWER_CTL     0x2D
#define DATAX0        0x32

const char codeVersion[3] = "0.2";  // code version number
const int timeDefault = 5;  // default duration of data stream, seconds
const int freqDefault = 5;  // default sampling rate of data stream, Hz
const int freqMax = 3200;  // maximal allowed cmdline arg sampling rate of data stream, Hz
const int speedSPI = 2000000;  // SPI communication speed, bps
const int freqMaxSPI = 100000;  // maximal possible communication sampling rate through SPI,
 Hz (assumption)
const int coldStartSamples = 2;  // number of samples to be read before outputting data to console
 (cold start delays)
const double coldStartDelay = 0.1;  // time delay between cold start reads
const double accConversion = 2 * 16.0 / 8192.0;  // +/- 16g range, 13-bit resolution
const double tStatusReport = 1;  // time period of status report if data read to file, seconds
void printUsage() {
    printf( "adxl345spi (version %s) \n"
            "License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>\n"
            "\n"
            "Usage: adxl345spi [OPTION]... \n"
            "Read data from ADXL345 accelerometer through SPI interface on Raspberry Pi.\n"
            "Online help, docs & bug reports: <https://github.com/nagimov/adxl345spi/>\n"
            "\n"
            "Mandatory arguments to long options are mandatory for short options too.\n"
            "  -s, --save FILE     save data to specified FILE (data printed to command-line\n"
            "                      output, if not specified)\n"
            "  -t, --time TIME     set the duration of data stream to TIME seconds\n"
            "                      (default: %i seconds) [integer]\n"
            "  -f, --freq FREQ     set the sampling rate of data stream to FREQ samples per\n"
            "                      second, 1 <= FREQ <= %i (default: %i Hz) [integer]\n"
            "\n"
            "Data is streamed in comma separated format, e. g.:\n"
            "  time,     x,     y,     z\n"
            "   0.0,  10.0,   0.0, -10.0\n"
            "   1.0,   5.0,  -5.0,  10.0\n"
            "   ...,   ...,   ...,   ...\n"
            "  time shows seconds elapsed since the first reading;\n"
            "  x, y and z show acceleration along x, y and z axis in fractions of <g>.\n"
            "\n"
            "Exit status:\n"
```

```
          "  0  if OK\n"
          "  1  if error occurred during data reading or wrong cmdline arguments.\n"
          "", codeVersion, timeDefault, freqMax, freqDefault);
}
int readBytes(int handle, char *data, int count) {
    data[0] |= READ_BIT;
    if (count > 1) data[0] |= MULTI_BIT;
    return spiXfer(handle, data, data, count);
}

int writeBytes(int handle, char *data, int count) {
    if (count > 1) data[0] |= MULTI_BIT;
    return spiWrite(handle, data, count);
}
int main(int argc, char *argv[]) {
    int i;
    // handling command-line arguments
    int bSave = 0;
    char vSave[256] = "";
    double vTime = timeDefault;
    double vFreq = freqDefault;
    for (i = 1; i < argc; i++) {  // skip argv[0] (program name)
        if ((strcmp(argv[i], "-s") == 0) || (strcmp(argv[i], "--save") == 0)) {
            bSave = 1;
            if (i + 1 <= argc - 1) {  // make sure there are enough arguments in argv
                i++;
                strcpy(vSave, argv[i]);
            }
            else {
                printUsage();
                return 1;
            }
        }
        else if ((strcmp(argv[i], "-t") == 0) || (strcmp(argv[i], "--time") == 0)) {
            if (i + 1 <= argc - 1) {
                i++;
                vTime = atoi(argv[i]);
            }
            else {
                printUsage();
                return 1;
            }
        }
        else if ((strcmp(argv[i], "-f") == 0) || (strcmp(argv[i], "--freq") == 0)) {
            if (i + 1 <= argc - 1) {
                i++;
                vFreq = atoi(argv[i]);
                if ((vFreq < 1) || (vFreq > 3200)) {
                    printf("Wrong sampling rate specified!\n\n");
                    printUsage();
                    return 1;
                }
            }
            else {
                printUsage();
                return 1;
            }
        }
        else {
            printUsage();
```

```
            return 1;
        }
    }
    // reading sensor data
    // SPI sensor setup
    int samples = vFreq * vTime;
    int samplesMaxSPI = freqMaxSPI * vTime;
    int success = 1;
    int h, bytes;
    char data[7];
    int16_t x, y, z;
    double tStart, tDuration, t;
    if (gpioInitialise() < 0) {
        printf("Failed to initialize GPIO!");
        return 1;
    }
    h = spiOpen(0, speedSPI, 3);
    data[0] = BW_RATE;
    data[1] = 0x0F;
    writeBytes(h, data, 2);
    data[0] = DATA_FORMAT;
    data[1] = DATA_FORMAT_B;
    writeBytes(h, data, 2);
    data[0] = POWER_CTL;
    data[1] = 0x08;
    writeBytes(h, data, 2);
    double delay = 1.0 / vFreq;  // delay between reads in seconds
// depending from the output mode (print to cmdline / save to file) data is read in
different ways
// for cmdline output, data is read directly to the screen with a sampling rate which is
*approximately* equal...
    // but always less than the specified value, since reading takes some time
    if (bSave == 0) {
        // fake reads to eliminate cold start timing issues (~0.01 s shift of sampling time
after the first reading)
        for (i = 0; i < coldStartSamples; i++) {
            data[0] = DATAX0;
            bytes = readBytes(h, data, 7);
            if (bytes != 7) {
                success = 0;
            }
            time_sleep(coldStartDelay);
        }
        // real reads happen here
        tStart = time_time();
        for (i = 0; i < samples; i++) {
            data[0] = DATAX0;
            bytes = readBytes(h, data, 7);
            if (bytes == 7) {
                x = (data[2]<<8)|data[1];
                y = (data[4]<<8)|data[3];
                z = (data[6]<<8)|data[5];
                t = time_time() - tStart;
                printf("time = %.3f, x = %.3f, y = %.3f, z = %.3f\n",
                       t, x * accConversion, y * accConversion, z * accConversion);
            }
            else {
                success = 0;
            }
            time_sleep(delay);  // pigpio sleep is accurate enough for console output, not
```

```
 necessary to use nanosleep
        }
        gpioTerminate();
        tDuration = time_time() - tStart;  // need to update current time to give a closer
estimate of sampling rate
        printf("%d samples read in %.2f seconds with sampling rate %.1f
 Hz\n", samples, tDuration, samples/tDuration);
        if (success == 0) {
            printf("Error occurred!");
            return 1;
        }
    }
    // for the file output, data is read with a maximal possible sampling rate (around
30,000 Hz)...
    // and then accurately rescaled to *exactly* match the specified sampling rate...
    // therefore, saved data can be easily analyzed (e. g. with fft)
    else {
        // reserve vectors for file-output arrays: time, x, y, z
        // arrays will not change their lengths, so separate track of the size is not needed
        double *at, *ax, *ay, *az;
        at = malloc(samples * sizeof(double));
        ax = malloc(samples * sizeof(double));
        ay = malloc(samples * sizeof(double));
        az = malloc(samples * sizeof(double));

        // reserve vectors for raw data: time, x, y, z
        // maximal achievable sampling rate depends from the hardware...
        // in my case, for Raspberry Pi 3 at 2 Mbps SPI bus speed sampling rate never
 exceeded ~30,000 Hz...
        // so to be sure that there is always enough memory allocated, freqMaxSPI is set
to 60,000 Hz
        double *rt, *rx, *ry, *rz;
        rt = malloc(samplesMaxSPI * sizeof(double));
        rx = malloc(samplesMaxSPI * sizeof(double));
        ry = malloc(samplesMaxSPI * sizeof(double));
        rz = malloc(samplesMaxSPI * sizeof(double));
        printf("Reading %d samples in %.1f seconds with sampling rate %.1f
Hz...\n", samples, vTime, vFreq);
        int statusReportedTimes = 0;
        double tCurrent, tClosest, tError, tErrorPrev, tLeft;
        int j, jClosest;
        tStart = time_time();
        int samplesRead;
        for (i = 0; i < samplesMaxSPI; i++) {
            data[0] = DATAX0;
            bytes = readBytes(h, data, 7);
            if (bytes == 7) {
                x = (data[2]<<8)|data[1];
                y = (data[4]<<8)|data[3];
                z = (data[6]<<8)|data[5];
                t = time_time();
                rx[i] = x * accConversion;
                ry[i] = y * accConversion;
                rz[i] = z * accConversion;
                rt[i] = t - tStart;
            }
            else {
                gpioTerminate();
                printf("Error occurred!");
                return 1;
```

```
            }
            tDuration = t - tStart;
            if (tDuration > tStatusReport * ((float)statusReportedTimes + 1.0)) {
                statusReportedTimes++;
                tLeft = vTime - tDuration;
                if (tLeft < 0) {
                    tLeft = 0.0;
                }
                printf("%.2f seconds left...\n", tLeft);
            }
            if (tDuration > vTime) {  // enough data read
                samplesRead = i;
                break;
            }
        }
        gpioTerminate();
        printf("Writing to the output file...\n");
        for (i = 0; i < samples; i++) {
            if (i == 0) {  // always get the first reading from position 0
                tCurrent = 0.0;
                jClosest = 0;
                tClosest = rt[jClosest];
            }
            else {
                tCurrent = (float)i * delay;
                tError = fabs(tClosest - tCurrent);
                tErrorPrev = tError;
                for (j = jClosest; j < samplesRead; j++) {  // lookup starting from previous j value
                    if (fabs(rt[j] - tCurrent) < tError) {  // in order to save some iterations
                        jClosest = j;
                        tClosest = rt[jClosest];
                        tErrorPrev = tError;
                        tError = fabs(tClosest - tCurrent);
                    }
                    else {
                        if (tError > tErrorPrev) {  // if the error starts growing
                            break;                  // break the loop since the minimal error point passed
                        }
                    }
                }  // when this loop is ended, jClosest and tClosest keep coordinates of the closest (j, t) point
            }
            ax[i] = rx[jClosest];
            ay[i] = ry[jClosest];
            az[i] = rz[jClosest];
            at[i] = tCurrent;
        }
        FILE *f;
        f = fopen(vSave, "w");
        fprintf(f, "time, x, y, z\n");
        for (i = 0; i < samples; i++) {
            fprintf(f, "%.5f, %.5f, %.5f, %.5f \n", at[i], ax[i], ay[i], az[i]);
        }
        fclose(f);
    }
    printf("Done\n");
    return 0;
}
```

8.4 A4 Vibration Analysis ADXL345 FFT Program

```
#### Prepare Modules #####

import matplotlib.pyplot as plt
import numpy as np
import scipy as scipy
import pandas as pd
import sys
import time
from numpy import NaN, Inf, arange, isscalar, asarray, array
from scipy import fft
from scipy import integrate
from scipy import signal

##### Only to use during testing - warnings have been reviewed carefully #####
import warnings
warnings.filterwarnings("ignore")

##### Setup Serial Connection #####
import serial
ser = serial.Serial('/dev/ttyACM0', 9600)
time.sleep(5)
ser.flushInput()

##### GPIO Output Setup #####
import RPi.GPIO as GPIO
GPIO.setmode(GPIO.BCM)
GPIO.setup(26, GPIO.OUT)        # Set GPIO26 as an output 'OK'
GPIO.setup(16, GPIO.OUT)        # Set GPIO16 as an output 'WE'
GPIO.setup(6, GPIO.OUT)         # Set GPIO6 as an output  'NO'
GPIO.setup(5, GPIO.OUT)         # Set GPIO5 as an output  'LE'
GPIO.setup(27, GPIO.OUT)        # Set GPIO27 as an output 'Life Bit to PLC'
GPIO.setup(22, GPIO.IN)        # Set GPIO22 as an input 'Life Bit from PLC'
GPIO.setup(17, GPIO.IN)        # Set GPIO17 as an input 'Press at bottom'



##### Open up the CSV file made by the other program #####


while True:
  if GPIO.input(22) and  not GPIO.input(17):
      GPIO.output(27,1)
      path = pd.read_csv(r'/home/pi/test19.csv')
      path = path.rename(columns={' x':'x'})
#Output from other program puts space in front of letter, needs correcting
      path = path.rename(columns={' y':'y'})
      path = path.rename(columns={' z':'z'})

      ##### Assign column data to Python Series #####

      t = path.time             # This looks at the time column in the CSV
      data = path.z             # This looks at the z-axis column in the CSV
      x = data - np.mean(data)     # Remove the DC offset from the data
      N = np.int(np.prod(t.shape)) #length of the array equals sample number
      Fs = 1/(t[1]-t[0])          #sample rate (Hz)
      T = 1/Fs;                  #time between samples
      #print("# Samples:",N)
      #print("Sample rate:",Fs,"Hz")
```

```
    #print(T)

    ##### Velocity Calculations  #####

    x_array = x.values
    x_conv = x_array * 9.81  # Convert the acceleration from g to m/s/s
    time_array = t.values
    vel2 = scipy.integrate.cumtrapz(x_conv,time_array) # in m/s
    vel3 = vel2 * 1000                          # in mm/s
    velrms = vel3 / 1.414213562373095              # in mm/s RMS

    ##### Plot Time Domain Based Acceleration Data #####
    #plt.ioff()    # Interactive plotting off when in auto mode, still saves
    plt.figure(1)
    plt.plot(t,data)  # Before DC offset is removed in blue
    plt.plot(t, x) # Plots the acceleration against the time (x,y)
    plt.xlabel('Time (seconds)')
    plt.ylabel('Accel (g)')
    plt.title("Time-domain Acceleration Plot")
    plt.grid()
    plt.show(block=False)
    plt.pause(0.0001)
    datetime = time.strftime("%Y%m%d-%H%M%S")
# Only save figures if enough memory free
    plt.savefig('Time Domain Based Acceleration Data -' + datetime + '.png')
# Only save figures if enough memory free

    ##### Plot Time Domain Based Velocity Data #####
    plt.figure(2)
    plt.plot(time_array[0:6399],vel3[0:6399]) #plotted in mm/s
    plt.plot(time_array[0:6399],velrms[0:6399]) #plotted in mm/s RMS
    plt.xlabel('Time (seconds)')
    plt.ylabel('Vel (mm/s)')
    plt.title("Time-domain Velocity Plot - showing drift")
    plt.grid()
    plt.show(block=False)
    plt.pause(0.0001)
    datetime = time.strftime("%Y%m%d-%H%M%S")
# Only save figures if enough memory free
    plt.savefig('Time Domain Based Velocity Data -' + datetime + '.png')
# Only save figures if enough memory free

      detrendednorm = scipy.signal.detrend(vel3)
      detrendedrms = scipy.signal.detrend(velrms)

      sos = scipy.signal.butter(10,0.0275,'hp', output='sos')
  #10th order, High pass Filter
      hpfiltrms = scipy.signal.sosfilt(sos,detrendedrms)
      hpfiltnorm = scipy.signal.sosfilt(sos,detrendednorm)

      plt.figure(3)
      plt.plot(time_array[0:6399],detrendednorm[0:6399])
      plt.plot(time_array[0:6399],detrendedrms[0:6399])
      plt.xlabel('Time (seconds)')
      plt.ylabel('Vel (mm/s)')
      plt.title("Time-domain Velocity Plot - detrended")
      plt.grid()
      plt.show(block=False)
      plt.pause(0.0001)
      datetime = time.strftime("%Y%m%d-%H%M%S")
```

```
# Only save figures if enough memory free
        plt.savefig('Time Domain Based Velocity Data - Detrended -' + datetime + '.png')
# Only save figures if enough memory free

        plt.figure(4)
        plt.plot(time_array[0:6399],hpfiltrms[0:6399])
        plt.xlabel('Time (seconds)')
        plt.ylabel('Vel (mm/s) RMS')
        plt.title("Time-domain RMS Velocity Plot - high pass filtered and detrended")
        plt.grid()
        plt.show(block=False)
        plt.pause(0.0001)
        datetime = time.strftime("%Y%m%d-%H%M%S")
# Only save figures if enough memory free
        plt.savefig('Time Domain Based Velocity Data - RMS Filtered and Detrended -' +
 datetime + '.png')
# Only save figures if enough memory free

        plt.figure(5)
        plt.plot(time_array[0:6399],hpfiltnorm[0:6399])
        plt.xlabel('Time (seconds)')
        plt.ylabel('Vel (mm/s)')
        plt.title("Time-domain Velocity Plot - high pass filtered and detrended")
        plt.grid()
        plt.show(block=False)
        plt.pause(0.0001)
        datetime = time.strftime("%Y%m%d-%H%M%S")
# Only save figures if enough memory free
        plt.savefig('Time Domain Based Velocity Data - Filtered and Detrended -' +
 datetime + '.png') # Only save figures if enough memory free

        ##### Compute and Plot Acceleration FFT #####

        # xf returns evenly spaced numbers over a specified sample
        # 0.0Hz is the start, 1.0/(2.0*T) Hz is the end point
        # N/2 is the number of samples evenly spaced
        # yff is the fft of the 'y' axis acceleration values
        # xf on the axis for frequencies
        # y axis acceleration values are normalised

        plt.figure(6)
        xf = np.linspace(0.0, 1.0/(2.0*T), N/2)
        yf = fft(x)
        yff = 2.0/N * np.abs(yf[0:np.int(N/2)])
        plt.plot(xf, yff)
        plt.xlim(0,400) #Can be up to 1600, adjust depending on graph shapes
        plt.grid()
        plt.xlabel('Frequency (Hz)')
        plt.ylabel('Accel (g)')
        plt.title('FFT of Acceleration Vibration Analysis')
        plt.show(block=False)
        plt.pause(0.0001)
        datetime = time.strftime("%Y%m%d-%H%M%S")
# Only save figures if enough memory free
        plt.savefig('Frequency Domain Based Acceleration Data -' + datetime + '.png')
# Only save figures if enough memory free

        ##### Compute and Plot Velocity FFT #####

        # xf is the same for both acceleration and velocity plots
```

```
        plt.figure(7)
        vyf = fft(hpfiltrms)
        vyff = 2.0/N * np.abs(vyf[0:np.int(N/2)])
        plt.plot(xf, vyff)
        plt.xlim(0,400)
        plt.grid()
        plt.xlabel('Frequency (Hz)')
        plt.ylabel('Velocity (mm/s RMS)')
        plt.title('FFT of RMS Velocity Vibration Analysis')
        plt.show(block=False)
        plt.pause(0.0001)
        datetime = time.strftime("%Y%m%d-%H%M%S")
# Only save figures if enough memory free
        plt.savefig('Frequency Domain Based RMS Velocity Data -' + datetime + '.png')
# Only save figures if enough memory free

        ##### Function for peak detect: https://gist.github.com/endolith/250860 #####

        def peakdet(v, delta, x = None):
            """
           Converted from MATLAB script at http://billauer.co.il/peakdet.html

           Returns two arrays

           function [maxtab, mintab]=peakdet(v, delta, x)
           %PEAKDET Detect peaks in a vector
           %        [MAXTAB, MINTAB] = PEAKDET(V, DELTA) finds the local
           %        maxima and minima ("peaks") in the vector V.
           %        MAXTAB and MINTAB consists of two columns. Column 1
           %        contains indices in V, and column 2 the found values.
           %
           %        With [MAXTAB, MINTAB] = PEAKDET(V, DELTA, X) the indices
           %        in MAXTAB and MINTAB are replaced with the corresponding
           %        X-values.
           %
           %        A point is considered a maximum peak if it has the maximal
           %        value, and was preceded (to the left) by a value lower by
           %        DELTA.

           % Eli Billauer, 3.4.05 (Explicitly not copyrighted).
           % This function is released to the public domain; Any use is allowed.

           """
            maxtab = []
            mintab = []

            if x is None:
                x = arange(len(v))

            v = asarray(v)

            if len(v) != len(x):
                sys.exit('Input vectors v and x must have same length')

            if not isscalar(delta):
                sys.exit('Input argument delta must be a scalar')

            if delta <= 0:
                sys.exit('Input argument delta must be positive')
```

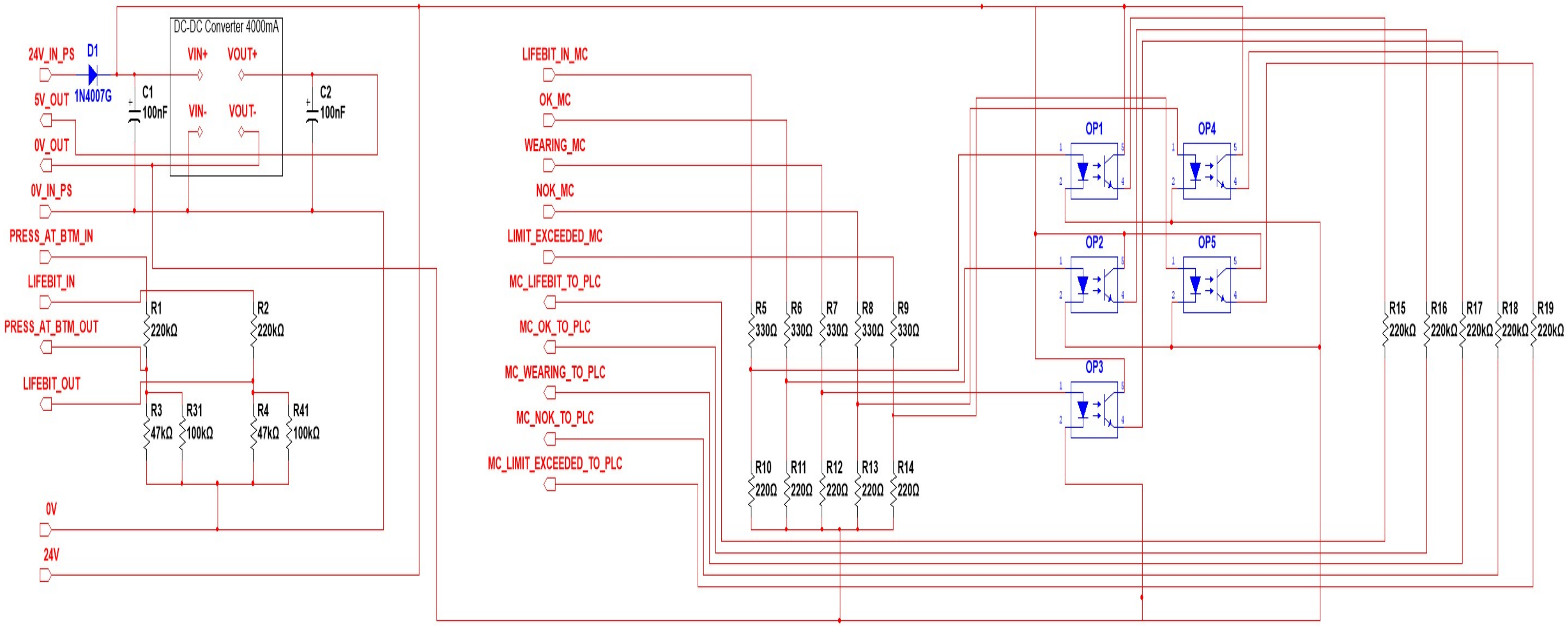
```python

        mn, mx = Inf, -Inf
        mnpos, mxpos = NaN, NaN

        lookformax = True

        for i in arange(len(v)):
            this = v[i]
            if this > mx:
                mx = this
                mxpos = x[i]
            if this < mn:
                mn = this
                mnpos = x[i]

            if lookformax:
                if this < mx-delta:
                    maxtab.append((mxpos, mx))
                    mn = this
                    mnpos = x[i]
                    lookformax = False
            else:
                if this > mn+delta:
                    mintab.append((mnpos, mn))
                    mx = this
                    mxpos = x[i]
                    lookformax = True

        return array(maxtab), array(mintab)


    ##### Peak Detect for Acceleration #####

    peaks = peakdet(yff,0.02,xf)
# 0.05 works for Test5 Z-axis, 0.04 works for Test5 X-axis, 0.01 works for z and x
    # with 0.01 you get more readings, but due to sorting, the erroneous ones are removed
    maxpeaks = (peaks[0])   # This looks at the maximum values only
    #print(maxpeaks)
    f1 = maxpeaks[0,0]      # Assign the first 4 identified values to an
    f2 = maxpeaks[1,0]      # area for easy sorting. Can be increased and
    f3 = maxpeaks[2,0]      # adjusted to personal preference.
    f4 = maxpeaks[3,0]
    g1 = maxpeaks[0,1]
    g2 = maxpeaks[1,1]
    g3 = maxpeaks[2,1]
    g4 = maxpeaks[3,1]

    sorts = [(f1,g1),(f2,g2),(f3,g3),(f4,g4)]
# List of Tuples to be sorted
    sorts.sort(key=lambda x: x[1],reverse = True)   # Sort by the g values
    peak1 = sorts[0]    # The sorts list is overwritten with the newly
    peak2 = sorts[1]
# sorted order from highest to lowest, the frequency
    peak3 = sorts[2]    # is moved alongside its acceleration
    peak4 = sorts[3]
    #print(sorts)
    print("There is a peak at",round((peak1[0]),3),"Hz with a g of",round((peak1[1]),3))
    print("There is a peak at",round((peak2[0]),3),"Hz with a g of",round((peak2[1]),3))
    print("There is a peak at",round((peak3[0]),3),"Hz with a g of",round((peak3[1]),3))
```

```
        print("There is a peak at",round((peak4[0]),3),"Hz with a g of",round((peak4[1]),3))
        ##### Peak Detect for Velocity #####

        vpeaks = peakdet(vyff,0.02,xf)
        vmaxpeaks = (vpeaks[0])
        #print(vmaxpeaks)
        vf1 = vmaxpeaks[0,0]     # Assign the first 3 identified values to an
        vf2 = vmaxpeaks[1,0]     # area for easy sorting. Can be increased and
        vf3 = vmaxpeaks[2,0]     # adjusted to personal preference.
        vf4 = vmaxpeaks[3,0]
        vg1 = vmaxpeaks[0,1]
        vg2 = vmaxpeaks[1,1]
        vg3 = vmaxpeaks[2,1]
        vg4 = vmaxpeaks[3,1]

        vsorts = [(vf1,vg1),(vf2,vg2),(vf3,vg3),(vf4,vg4)]
# List of Tuples to be sorted
        vsorts.sort(key=lambda x: x[1],reverse = True)     # Sort by the g values
        vpeak1 = vsorts[0]    # The sorts list is overwritten with the newly
        vpeak2 = vsorts[1]
# sorted order from highest to lowest, the frequency
        vpeak3 = vsorts[2]    # is moved alongside its acceleration
        vpeak4 = vsorts[3]
        #print(vsorts)
        print("There is a peak at",round((vpeak1[0]),3),"Hz with a mm/s RMS velocity
 of",round((vpeak1[1]),3))
        print("There is a peak at",round((vpeak2[0]),3),"Hz with a mm/s RMS velocity
 of",round((vpeak2[1]),3))
        print("There is a peak at",round((vpeak3[0]),3),"Hz with a mm/s RMS velocity
 of",round((vpeak3[1]),3))
        print("There is a peak at",round((vpeak4[0]),3),"Hz with a mm/s RMS velocity
 of",round((vpeak4[1]),3))

        ##### Warnings giving outputs to Arduino #####

        ##### Can be customised, can do velocity or acceleration #####
        ##### or can alter the program to incorporate both #####
        ##### Ideally, do one or the other, then manually swap #####



        ##### Mechanical Looseness #####

        a_looseness = (0.2 * (peak1[1]))
#[0] is frequency, [1] is amplitude
        v_looseness = (0.2 * (vpeak1[1]))

        if peak2[1] > a_looseness:
            if peak3[1] > a_looseness:
                if peak4[1] > a_looseness:
                    print("")
                    print("There is mechanical looseness.")
                    print("")
                    print("This equipment is wearing.")
                    print("")
                    ser.write(b'2') #1=OK,2=WE,3=NO,4=LE,5=ER
                    time.sleep(1)
                    GPIO.output(26,0) #OK
                    GPIO.output(16,1) #WE
                    GPIO.output(6,0)  #NO
```

```
            GPIO.output(5,0)  #LE

##### Misalignment #####

a_misalignment_lb = 0.5 * peak1[1]
a_misalignment_hb = 1.5 * peak1[1]
v_misalignment_lb = 0.5 * vpeak1[1]
v_misalignment_hb = 1.5 * vpeak1[1]

if peak2[1] > a_misalignment_lb and peak2[1] < a_misalignment_hb:
    print("")
    print("There is misalignment.")
    print("")
    print("This equipment is wearing.")
    print("")
    ser.write(b'2') #1=OK,2=WE,3=NO,4=LE,5=ER
    time.sleep(1)
    GPIO.output(26,0) #OK
    GPIO.output(16,1) #WE
    GPIO.output(6,0)  #NO
    GPIO.output(5,0)  #LE

if peak2[1] >a_misalignment_hb:
    print("")
    print("There is misalignment.")
    print("")
    print("This equipment is not OK.")
    print("")
    ser.write(b'3') #1=OK,2=WE,3=NO,4=LE,5=ER
    time.sleep(1)
    GPIO.output(26,0) #OK
    GPIO.output(16,0) #WE
    GPIO.output(6,1)  #NO
    GPIO.output(5,0)  #LE


##### Unbalance #####

if peak1[1] > 1 and peak1[1] < 10:
    print("")
    print("There is unbalanace.")
    print("")
    print("This equipment is wearing.")
    print("")
    ser.write(b'2') #1=OK,2=WE,3=NO,4=LE,5=ER
    time.sleep(1)
    GPIO.output(26,0) #OK
    GPIO.output(16,1) #WE
    GPIO.output(6,0)  #NO
    GPIO.output(5,0)  #LE


##### Good Running Condition #####

if peak1[1] < 1:
    if peak2[1] < a_looseness:
        if peak3[1] < a_looseness:
            if peak4[1] < a_looseness:
                print("")
                print("The equipment is running well.")
```

```
                    print("")
                    print("This equipment is OK.")
                    print("")
                    ser.write(b'1') #1=OK,2=WE,3=NO,4=LE,5=ER
                    time.sleep(1)
                    GPIO.output(26,1) #OK
                    GPIO.output(16,0) #WE
                    GPIO.output(6,0)  #NO
                    GPIO.output(5,0)  #LE


    ##### Limit Exceeded Condition #####

    if peak1[0] > 1500 or peak1[1] > 15:
        print("")
        print("The limits have been exceeded.")
        print("")
        print("The limit is exceeded.")
        print("")
        ser.write(b'4') #1=OK,2=WE,3=NO,4=LE,5=ER
        time.sleep(1)
        GPIO.output(26,0) #OK
        GPIO.output(16,0) #WE
        GPIO.output(6,0)  #NO
        GPIO.output(5,1)  #LE


    ##### Error Condition #####

    if peak1[1] == 0:
        print("")
        print("The limits have been exceeded.")
        print("")
        print("The limit is exceeded.")
        print("")
        ser.write(b'5') #1=OK,2=WE,3=NO,4=LE,5=ER
        time.sleep(1)
        GPIO.output(26,0) #OK
        GPIO.output(16,0) #WE
        GPIO.output(6,0)  #NO
        GPIO.output(5,0)  #LE
        # Error is not a GPIO only a CGD warning

    ##### The above warnings suit the test program and each motor will have
    ##### its own adjusted settings for optimum accuracy.

    time.sleep(60) # Loop the program again after (x) seconds
    GPIO.output(27,0)
    GPIO.output(26,0) #OK
    GPIO.output(16,0) #WE
    GPIO.output(6,0)  #NO
    GPIO.output(5,0)  #LE

else:
    print("PLC Conditions not met!")
    #ser.write(b'5') #1=OK,2=WE,3=NO,4=LE,5=ER
    time.sleep(10)
```

8.5 A5 Vibration Analysis Interfacing Circuit Schematic



8.6 A6 Non-Intrusive Pipe Pressure Arduino Program excluding nRF

```
/* Input and Output Declarations */

#define strain A0               // Analog input from the Strain Gauge sensor

const int calib = 30;           // Calibration button input to set a datum when pumps
                                   are off - hold for 5 seconds;
const int pumpson = 31;         // Pumps running signal from the PLC
const int lifebitin = 34;       // Life Bit signal from the PLC
const int lifebitout = 35;      // Life Bit signal to the PLC
const int calibcomp = 36;       // Calibration complete to allow measurement
const int bracketone = 32;      // Current condition is 0 to 10 bar signal to the PLC
const int brackettwo = 33;      // Current condition is 10 to 20 bar signal to the PLC
const int bracketthree = 37;    // Current condition is 20 to 30 bar signal to the PLC
const int bracketfour = 38;     // Current condition is 30 to 40 bar signal to the PLC
const int bracketfive = 39;     // Current condition is 40 to 50 bar signal to the PLC
const int bracketsix = 40;      // Current condition is 50 to 60 bar signal to the PLC
const int bracketseven = 41;    // Current condition is 60 to 70 bar signal to the PLC
const int bracketeight = 42;    // Current condition is 70 to 80 bar signal to the PLC
const int bracketnine = 43;     // Current condition is 80 to 90 bar signal to the PLC
const int bracketten = 44;      // Current condition is 90 to 100 bar signal to the PLC
const int LimitExceeded = 45;   // The boundary limits have been exceeded signal to the
                                   PLC

/* Setup Local Variables */

float reading0 = 0.000;         // First hold area for the first reading
float reading1 = 0.000;         // Second hold area for the second reading
float reading2 = 0.000;         // Third hold area for the third reading
float reading3 = 0.000;         // Fourth hold area for the third reading
float reading4 = 0.000;         // Fifth hold area for the third reading
float reading5 = 0.000;         // Sixth hold area for the third reading
float reading6 = 0.000;         // Seventh hold area for the third reading
float reading7 = 0.000;         // Eighth hold area for the third reading
float reading8 = 0.000;         // Ninth hold area for the third reading
float reading9 = 0.000;         // Tenth hold area for the third reading
float checkreading;             // Check each reading before confirmation
float avgreading;               // Average of all the averages
float calibreading;             // The initial calibration reading at 0 bar pumps off
float avgone;                   // The first average from the first ten readings
float avgtwo;                   // The second average from the second ten readings
float avgthree;                 // The third average from the third ten readings

int calibcompinternal;          // Internal bit for calibration completion checking
int avgreadingfirstcomp;        // Checks the first avg reading has been taken


void setup() {

  pinMode(calib, INPUT);                // Set calib pin as an input
  pinMode(pumpson, INPUT);              // Set presstdc pin as an input
  pinMode(lifebitin, INPUT);            // Set lifebitin pin as an input
  pinMode(lifebitout, OUTPUT);          // Set lifebitout pin as an output
  pinMode(bracketone, OUTPUT);          // Set bracketone pin as an output
  pinMode(brackettwo, OUTPUT);          // Set brackettwo pin as an output
  pinMode(bracketthree, OUTPUT);        // Set bracketthree pin as an output
  pinMode(bracketfour, OUTPUT);         // Set bracketfour pin as an output
  pinMode(bracketfive, OUTPUT);         // Set bracketfive pin as an output
  pinMode(bracketsix, OUTPUT);          // Set bracketsix pin as an output
```

```
pinMode(bracketseven, OUTPUT);       // Set bracketseven pin as an output
pinMode(bracketeight, OUTPUT);       // Set bracketeight pin as an output
pinMode(bracketnine, OUTPUT);        // Set bracketnine pin as an output
pinMode(bracketten, OUTPUT);         // Set bracketten pin as an output
pinMode(calibcomp, OUTPUT);          // Set calibcomp pin as an output
pinMode(LimitExceeded, OUTPUT);      // Set LimitExceeded as an output
digitalWrite(calibcomp, LOW);        // Set calibcomp low on power cycle

          }

void loop() {

  if (digitalRead(lifebitin) == HIGH){
// Check lifebit from PLC is OK

   digitalWrite(lifebitout, HIGH);
// Send lifebit to PLC

   if (digitalRead(pumpson) == LOW)                {
// Check that the pumps are OFF for 0 bar calibration

     if (digitalRead(calib) == HIGH){
// Check calibration button is pressed

       float measurement = analogRead(strain);
// Read the value from the strain sensor circuit
       float measasvolt = measurement * 0.0048828125;
// Convert value from sensor to a voltage
       calibreading =  (((measasvolt)-0.25)/(-0.0018));
// Use the derived equation to convert to a pressure
       delay(500);
// Pause 0.5s
       digitalWrite(calibcomp, HIGH);
// Set calibcomp signal high to PLC
       calibcompinternal = HIGH;
// Set internal calibration complete signal high
                              }
       delay(100);
 // Pause 0.1s
                                             }

     if (digitalRead(pumpson) == HIGH)               {
  // Only allow for readings if pumps are on

      if (calibcompinternal == HIGH){
    // Only allow for reading to take place if calibrated
       for (int j = 0; j<=2; j++){
        for (int i = 0; i <=9; i++) {
       // Initialise For Loop for 3 readings
         chk:
          float measurement = analogRead(strain);
    // Read the value from the sensor
          float measasvolt = measurement * 0.0048828125;
  // Convert value from sensor to a voltage
          float checkreading = (((measasvolt)-0.25)/(-0.0018))- calibreading;
          delay(100);
            if (checkreading > 150){
              goto chk;
              delay(100);        }

```

```
        if (i == 0)     {
    // The first reading
            reading0 = (((measasvolt)-0.25)/(-0.0018)) - calibreading;
    // Find the pressure and offset the 0 bar calibration
            delay(10);    }
    // Pause 0.01s

        if (i == 1)     {
     // The second reading
            reading1 = (((measasvolt)-0.25)/(-0.0018)) - calibreading;
   // Find the pressure and offset the 0 bar calibration
            delay(10);    }
      // Pause 0.01s

        if (i == 2)     {
     // The third reading
            reading2 = (((measasvolt)-0.25)/(-0.0018)) - calibreading;
   // Find the pressure and offset the 0 bar calibration
            delay(10);    }
     // Pause 0.01s

        if (i == 3)     {
    // The fourth reading
            reading3 = (((measasvolt)-0.25)/(-0.0018)) - calibreading;
     // Find the pressure and offset the 0 bar calibration
            delay(10);    }
      // Pause 0.01s

        if (i == 4)     {
      // The fifth reading
            reading4 = (((measasvolt)-0.25)/(-0.0018)) - calibreading;
      // Find the pressure and offset the 0 bar calibration
            delay(10);    }
      // Pause 0.01s

        if (i == 5)     {
     // The sixth reading
            reading5 = (((measasvolt)-0.25)/(-0.0018)) - calibreading;
     // Find the pressure and offset the 0 bar calibration
            delay(10);    }
      // Pause 0.01s

        if (i == 6)     {
      // The seventh reading
            reading6 = (((measasvolt)-0.25)/(-0.0018)) - calibreading;
     // Find the pressure and offset the 0 bar calibration
            delay(10);    }
      // Pause 0.01s

        if (i == 7)     {
    // The eighth reading
            reading7 = (((measasvolt)-0.25)/(-0.0018)) - calibreading;
    // Find the pressure and offset the 0 bar calibration
            delay(10);    }
     // Pause 0.01s

        if (i == 8)     {
     // The ninth reading
            reading8 = (((measasvolt)-0.25)/(-0.0018)) - calibreading;
     // Find the pressure and offset the 0 bar calibration
```

```
        delay(10);   }
    // Pause 0.01s
      if (i == 9)     {
  // The tenth reading
        reading9 = (((measasvolt)-0.25)/(-0.0018)) - calibreading;
   // Find the pressure and offset the 0 bar calibration
        delay(10);   }
  // Pause 0.01s
                    }
      avgreading = ((reading0 + reading1 + reading2 + reading3 + reading4 +
reading5 + reading6 + reading7 + reading8 + reading9)/10);
  // Average the readings after 10 measurements
      avgreadingfirstcomp = HIGH;
      if (j == 0){
 // The first set of averages
      avgone = avgreading; }
 // The current average difference is copied to avgone hold
      delay(50);
      if (j == 1){
 // The second set of averages
      avgtwo = avgreading; }
  // The current average difference is copied to avgtwo hold
      delay(50);
      if (j == 2){
  // The third set of averages
      avgthree = avgreading;}
   // The current average difference is copied to avgthree hold
      delay(50);
                  }
     if(avgreadingfirstcomp == HIGH){
  // Check that a read has been calculated before output
      if (avgone >= 0 && avgone < 10 && avgtwo >= 0 && avgtwo < 10 &&
avgthree >= 0 && avgthree < 10){
 // Check all average differences in a set range for 0 to 10 bar
     digitalWrite(bracketone,   HIGH);
  // Measurement is in Bracket 1
     digitalWrite(brackettwo,   LOW);
 // Measurement is not in Bracket 2
     digitalWrite(bracketthree,  LOW);
   // Measurement is not in Bracket 3
     digitalWrite(bracketfour,   LOW);
    // Measurement is not in Bracket 4
     digitalWrite(bracketfive,   LOW);
       // Measurement is not in Bracket 5
     digitalWrite(bracketsix,    LOW);
    // Measurement is not in Bracket 6
     digitalWrite(bracketseven,  LOW);
    // Measurement is not in Bracket 7
     digitalWrite(bracketeight,  LOW);
   // Measurement is not in Bracket 8
     digitalWrite(bracketnine,   LOW);
   // Measurement is not in Bracket 9
     digitalWrite(bracketten,    LOW);
  // Measurement is not in Bracket 10
     digitalWrite(LimitExceeded, LOW);
```

```
 // The limits have not been exceeded
}
    else if (avgone >= 10 && avgone < 20 && avgtwo >= 10 && avgtwo < 20 &&
avgthree >= 10 && avgthree < 20){
// Check all average differences in a set range for 10 to 20 bar

      digitalWrite(bracketone,    LOW);
// Measurement is not in Bracket 1
      digitalWrite(brackettwo,    HIGH);
// Measurement is in Bracket 2
      digitalWrite(bracketthree,  LOW);
  // Measurement is not in Bracket 3
      digitalWrite(bracketfour,   LOW);
// Measurement is not in Bracket 4
      digitalWrite(bracketfive,   LOW);
// Measurement is not in Bracket 5
      digitalWrite(bracketsix,    LOW);
// Measurement is not in Bracket 6
      digitalWrite(bracketseven,  LOW);
 // Measurement is not in Bracket 7
      digitalWrite(bracketeight,  LOW);
// Measurement is not in Bracket 8
      digitalWrite(bracketnine,   LOW);
 // Measurement is not in Bracket 9
      digitalWrite(bracketten,    LOW);
// Measurement is not in Bracket 10
      digitalWrite(LimitExceeded, LOW);
// The limits have not been exceeded
}
    else if (avgone >= 20 && avgone < 30 && avgtwo >= 20 && avgtwo < 30 &&
avgthree >= 20 && avgthree < 30){
// Check all average differences in a set range for 20 to 30 bar

      digitalWrite(bracketone,    LOW);
   // Measurement is not in Bracket 1
      digitalWrite(brackettwo,    LOW);
 // Measurement is not in Bracket 2
      digitalWrite(bracketthree,  HIGH);
 // Measurement is in Bracket 3
      digitalWrite(bracketfour,   LOW);
 // Measurement is not in Bracket 4
      digitalWrite(bracketfive,   LOW);
  // Measurement is not in Bracket 5
      digitalWrite(bracketsix,    LOW);
  // Measurement is not in Bracket 6
      digitalWrite(bracketseven,  LOW);
  // Measurement is not in Bracket 7
      digitalWrite(bracketeight,  LOW);
  // Measurement is not in Bracket 8
      digitalWrite(bracketnine,   LOW);
  // Measurement is not in Bracket 9
      digitalWrite(bracketten,    LOW);
  // Measurement is not in Bracket 10
      digitalWrite(LimitExceeded, LOW);
   // The limits have not been exceeded
 }
    else if (avgone >= 30 && avgone < 40 && avgtwo >= 30 && avgtwo < 40 &&
avgthree >= 30 && avgthree < 40){
// Check all average differences in a set range for 30 to 40 bar

```

```
        digitalWrite(bracketone,    LOW);
    // Measurement is not in Bracket 1
        digitalWrite(brackettwo,    LOW);
    // Measurement is not in Bracket 2
        digitalWrite(bracketthree,  LOW);
    // Measurement is not in Bracket 3
        digitalWrite(bracketfour,   HIGH);
    // Measurement is in Bracket 4
        digitalWrite(bracketfive,   LOW);
    // Measurement is not in Bracket 5
        digitalWrite(bracketsix,    LOW);
    // Measurement is not in Bracket 6

        digitalWrite(bracketseven,  LOW);
     // Measurement is not in Bracket 7
        digitalWrite(bracketeight,  LOW);
     // Measurement is not in Bracket 8
        digitalWrite(bracketnine,   LOW);
      // Measurement is not in Bracket 9
        digitalWrite(bracketten,    LOW);
     // Measurement is not in Bracket 10
        digitalWrite(LimitExceeded, LOW);
    // The limits have not been exceeded

         }
       else if (avgone >= 40 && avgone < 50 && avgtwo >= 40 && avgtwo <
    50 && avgthree >= 40 && avgthree < 50){
   // Check all average differences in a set range for 40 to 50 bar

        digitalWrite(bracketone,    LOW);
     // Measurement is not in Bracket 1
        digitalWrite(brackettwo,    LOW);
     // Measurement is not in Bracket 2
        digitalWrite(bracketthree,  LOW);
    // Measurement is not in Bracket 3
        digitalWrite(bracketfour,   LOW);
    // Measurement is not in Bracket 4
        digitalWrite(bracketfive,   HIGH);
    // Measurement is in Bracket 5
        digitalWrite(bracketsix,    LOW);
    // Measurement is not in Bracket 6
        digitalWrite(bracketseven,  LOW);
     // Measurement is not in Bracket 7
        digitalWrite(bracketeight,  LOW);
     // Measurement is not in Bracket 8
        digitalWrite(bracketnine,   LOW);
    // Measurement is not in Bracket 9
        digitalWrite(bracketten,    LOW);
      // Measurement is not in Bracket 10
        digitalWrite(LimitExceeded, LOW);
      // The limits have not been exceeded
      }

       else if (avgone >= 50 && avgone < 60 && avgtwo >= 50 && avgtwo < 60 &&
   avgthree >= 50 && avgthree < 60){
   // Check all average differences in a set range for 50 to 60 bar

        digitalWrite(bracketone,    LOW);
    // Measurement is not in Bracket 1
        digitalWrite(brackettwo,    LOW);
```

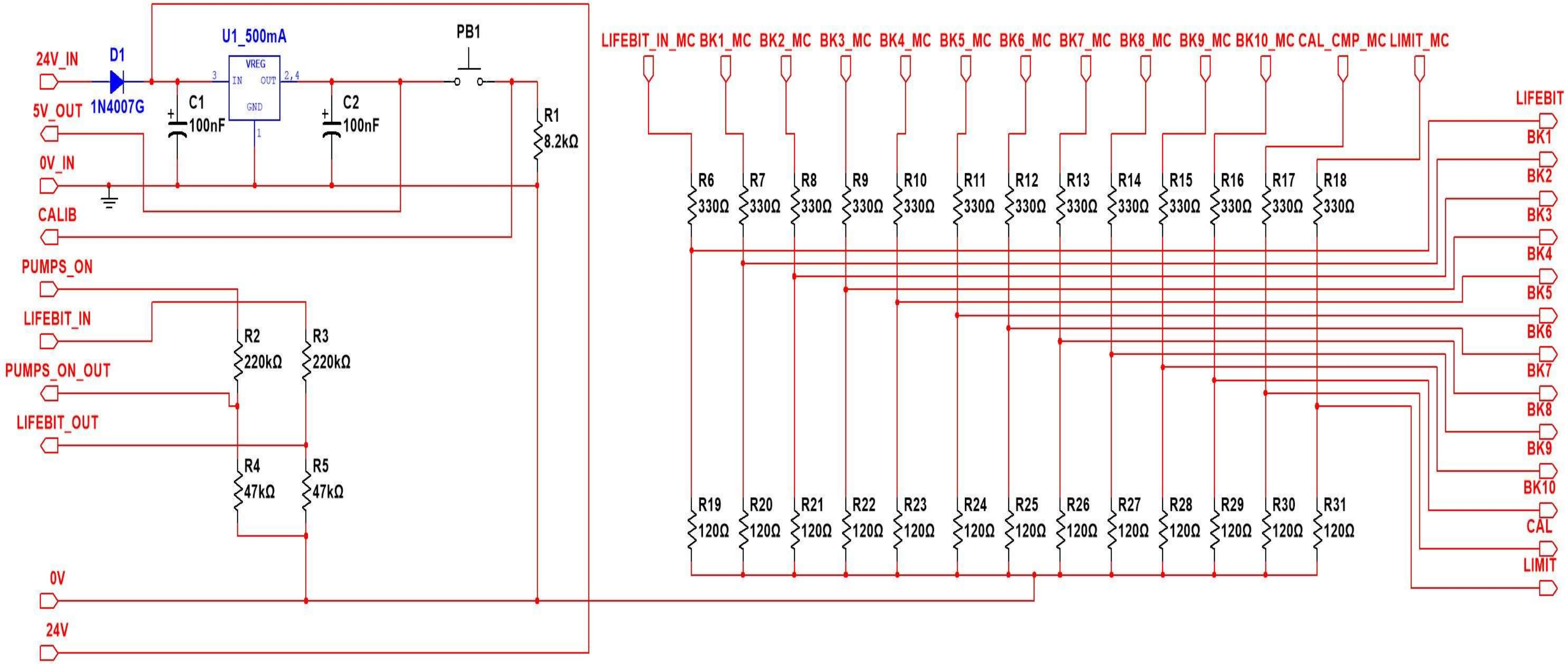
```
   // Measurement is not in Bracket 2
        digitalWrite(bracketthree,  LOW);
    // Measurement is not in Bracket 3
        digitalWrite(bracketfour,   LOW);
    // Measurement is not in Bracket 4
        digitalWrite(bracketfive,   LOW);
    // Measurement is not in Bracket 5
        digitalWrite(bracketsix,    HIGH);
    // Measurement is in Bracket 6
        digitalWrite(bracketseven,  LOW);
   // Measurement is not in Bracket 7
        digitalWrite(bracketeight,  LOW);
    // Measurement is not in Bracket 8
        digitalWrite(bracketnine,   LOW);
   // Measurement is not in Bracket 9
        digitalWrite(bracketten,    LOW);
   // Measurement is not in Bracket 10
        digitalWrite(LimitExceeded, LOW);
   // The limits have not been exceeded
   }

       else if (avgone >= 60 && avgone < 70 && avgtwo >= 60 && avgtwo < 70 &&
  avgthree >= 60 && avgthree < 70){
  // Check all average differences in a set range for 60 to 70 bar

        digitalWrite(bracketone,    LOW);
    // Measurement is not in Bracket 1
        digitalWrite(brackettwo,    LOW);
    // Measurement is not in Bracket 2
        digitalWrite(bracketthree,  LOW);
    // Measurement is not in Bracket 3
        digitalWrite(bracketfour,   LOW);
    // Measurement is not in Bracket 4
        digitalWrite(bracketfive,   LOW);
  // Measurement is not in Bracket 5
        digitalWrite(bracketsix,    LOW);
   // Measurement is not in Bracket 6
        digitalWrite(bracketseven,  HIGH);
    // Measurement is in Bracket 7
        digitalWrite(bracketeight,  LOW);
    // Measurement is not in Bracket 8
        digitalWrite(bracketnine,   LOW);
    // Measurement is not in Bracket 9
        digitalWrite(bracketten,    LOW);
    // Measurement is not in Bracket 10
        digitalWrite(LimitExceeded, LOW);
    // The limits have not been exceeded
   }

       else if (avgone >= 70 && avgone < 80 && avgtwo >= 70 && avgtwo < 80 &&
  avgthree >= 70 && avgthree < 80){
  // Check all average differences in a set range for 70 to 80 bar

        digitalWrite(bracketone,    LOW);
   // Measurement is not in Bracket 1
        digitalWrite(brackettwo,    LOW);
    // Measurement is not in Bracket 2
        digitalWrite(bracketthree,  LOW);
   // Measurement is not in Bracket 3
        digitalWrite(bracketfour,   LOW);
```

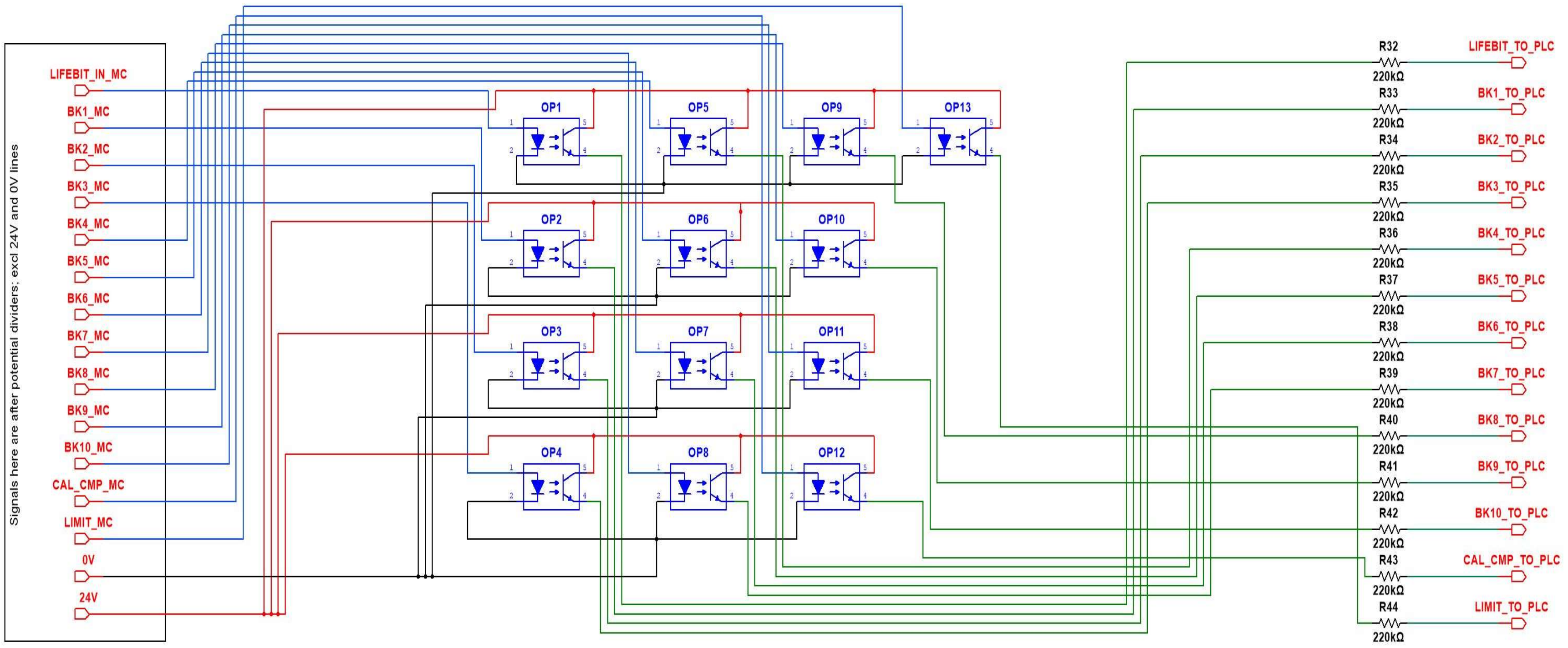
```
// Measurement is not in Bracket 4
      digitalWrite(bracketfive,   LOW);
 // Measurement is not in Bracket 5
      digitalWrite(bracketsix,    LOW);
 // Measurement is not in Bracket 6
      digitalWrite(bracketseven,  LOW);
  // Measurement is not in Bracket 7
      digitalWrite(bracketeight,  HIGH);
  // Measurement is in Bracket 8
      digitalWrite(bracketnine,   LOW);
  // Measurement is not in Bracket 9
      digitalWrite(bracketten,    LOW);
   // Measurement is not in Bracket 10
      digitalWrite(LimitExceeded, LOW);
   // The limits have not been exceeded

           }

     else if (avgone >= 80 && avgone < 90 && avgtwo >= 80 && avgtwo < 90 &&
avgthree >= 80 && avgthree < 90){
// Check all average differences in a set range for 80 to 90 bar

      digitalWrite(bracketone,    LOW);
   // Measurement is not in Bracket 1
      digitalWrite(brackettwo,    LOW);
   // Measurement is not in Bracket 2
      digitalWrite(bracketthree,  LOW);
  // Measurement is not in Bracket 3
      digitalWrite(bracketfour,   LOW);
    // Measurement is not in Bracket 4
      digitalWrite(bracketfive,   LOW);
    // Measurement is not in Bracket 5
      digitalWrite(bracketsix,    LOW);
     // Measurement is not in Bracket 6
      digitalWrite(bracketseven,  LOW);
      // Measurement is not in Bracket 7
      digitalWrite(bracketeight,  LOW);
      // Measurement is not in Bracket 8
      digitalWrite(bracketnine,   HIGH);
       // Measurement is in Bracket 9
      digitalWrite(bracketten,    LOW);
  // Measurement is not in Bracket 10
      digitalWrite(LimitExceeded, LOW);
     // The limits have not been exceeded
 }

     else if (avgone >= 90 && avgone < 100 && avgtwo >= 90 && avgtwo < 100 &&
 avgthree >= 90 && avgthree < 100){
// Check all average differences in a set range for 90 to 100 bar

      digitalWrite(bracketone,    LOW);
  // Measurement is not in Bracket 1
      digitalWrite(brackettwo,    LOW);
   // Measurement is not in Bracket 2
      digitalWrite(bracketthree,  LOW);
  // Measurement is not in Bracket 3
      digitalWrite(bracketfour,   LOW);
     // Measurement is not in Bracket 4
      digitalWrite(bracketfive,   LOW);
 // Measurement is not in Bracket 5
```

```
digitalWrite(bracketsix,    LOW);
// Measurement is not in Bracket 6
digitalWrite(bracketseven,  LOW);
// Measurement is not in Bracket 7
digitalWrite(bracketeight,  LOW);
// Measurement is not in Bracket 8
digitalWrite(bracketnine,   LOW);
// Measurement is not in Bracket 9
digitalWrite(bracketten,    HIGH);
// Measurement is in Bracket 10
digitalWrite(LimitExceeded, LOW);
// The limits have not been exceeded
}
else if (avgone < 0 && avgtwo < 0 && avgthree < 0){
digitalWrite(bracketone,    LOW);
// Measurement is not in Bracket 1
digitalWrite(brackettwo,    LOW);
// Measurement is not in Bracket 2
digitalWrite(bracketthree,  LOW);
// Measurement is not in Bracket 3
digitalWrite(bracketfour,   LOW);
// Measurement is not in Bracket 4
digitalWrite(bracketfive,   LOW);
// Measurement is not in Bracket 5
digitalWrite(bracketsix,    LOW);
// Measurement is not in Bracket 6
digitalWrite(bracketseven,  LOW);
// Measurement is not in Bracket 7
digitalWrite(bracketeight,  LOW);
// Measurement is not in Bracket 8
digitalWrite(bracketnine,   LOW);
// Measurement is not in Bracket 9
digitalWrite(bracketten,    LOW);
// Measurement is not in Bracket 10
digitalWrite(LimitExceeded, LOW);
// The limits have been exceeded but it is classified as an error

                              }

else if (avgone > 100 && avgtwo > 100 && avgthree > 100){
digitalWrite(bracketone,    LOW);
// Measurement is not in Bracket 1
digitalWrite(brackettwo,    LOW);
// Measurement is not in Bracket 2
digitalWrite(bracketthree,  LOW);
// Measurement is not in Bracket 3
digitalWrite(bracketfour,   LOW);
// Measurement is not in Bracket 4
digitalWrite(bracketfive,   LOW);
// Measurement is not in Bracket 5
digitalWrite(bracketsix,    LOW);
// Measurement is not in Bracket 6
digitalWrite(bracketseven,  LOW);
// Measurement is not in Bracket 7
digitalWrite(bracketeight,  LOW);
// Measurement is not in Bracket 8
digitalWrite(bracketnine,   LOW);
// Measurement is not in Bracket 9
digitalWrite(bracketten,    LOW);
// Measurement is not in Bracket 10
```

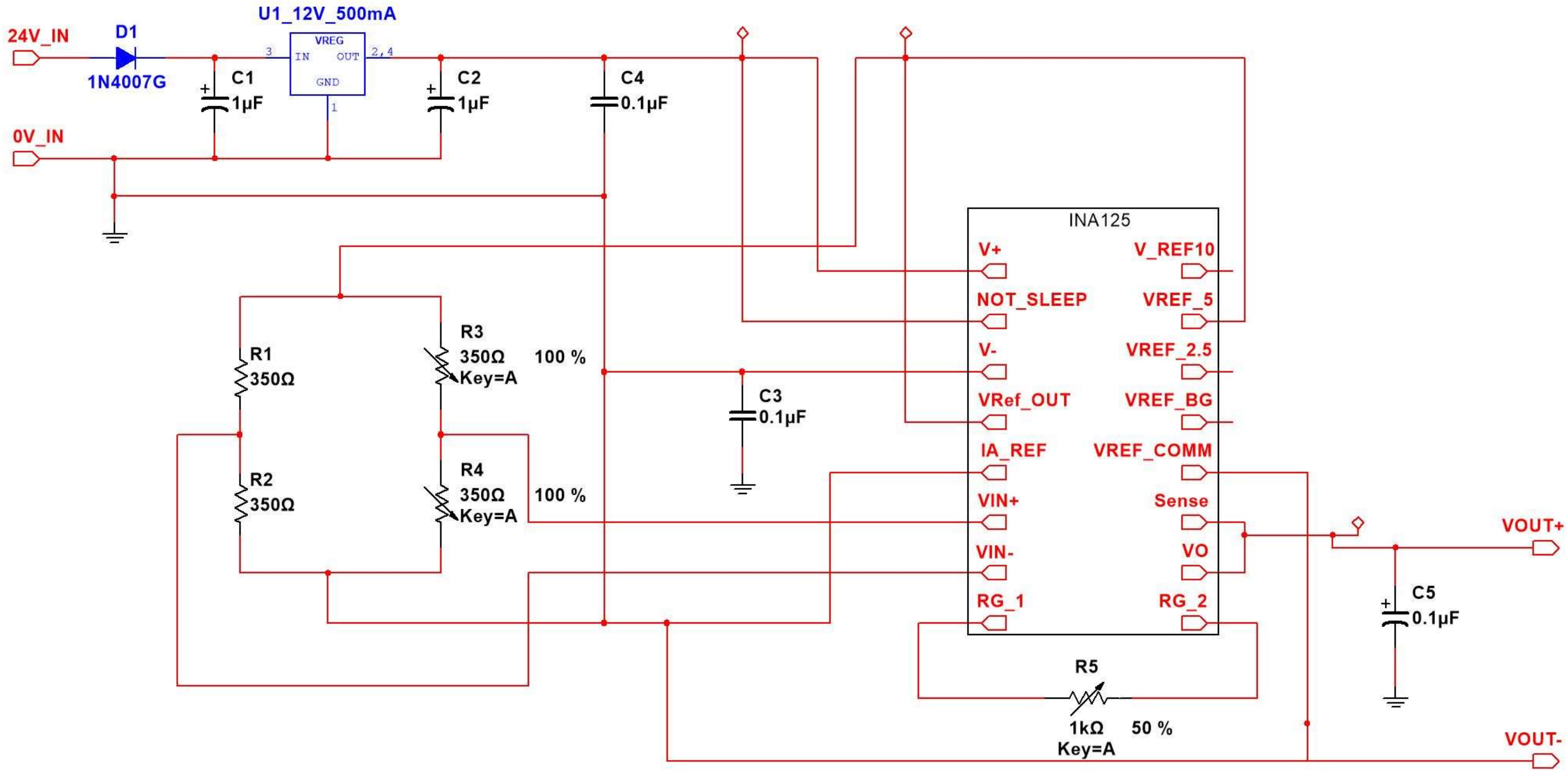
```
         digitalWrite(LimitExceeded, HIGH);
// The limits have been exceeded

            }

        delay(1000);
// Delay 1s
        digitalWrite(lifebitout, LOW);
// Reset lifebit to PLC before next loop
        }
        }
        }
        }
        }
```

8.7 A7 Non-Intrusive Pipe Pressure Interfacing and Calibration Circuit

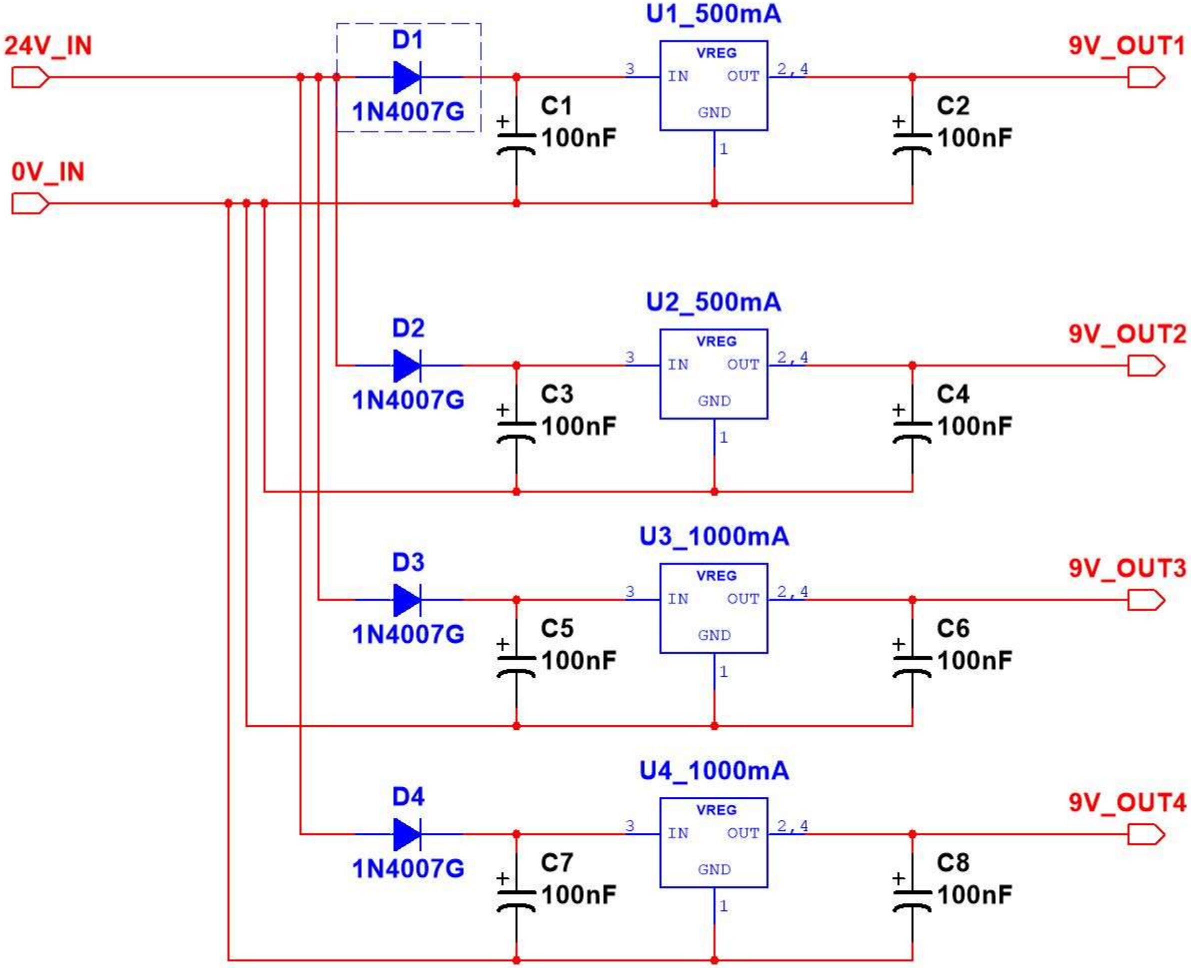



Signals here are after potential dividers; excl 24V and 0V lines
LIFEBIT_IN_MC
BK1_MC
BK2_MC
BK3_MC
BK4_MC
BK5_MC
BK6_MC
BK7_MC
BK8_MC
BK9_MC
BK10_MC
CAL_CMP_MC
LIMIT_MC
0V
24V
OP1
OP2
OP3
OP4
OP5
OP6
OP7
OP8
OP9
OP10
OP11
OP12
OP13
R32 220kΩ LIFEBIT_TO_PLC
R33 220kΩ BK1_TO_PLC
R34 220kΩ BK2_TO_PLC
R35 220kΩ BK3_TO_PLC
R36 220kΩ BK4_TO_PLC
R37 220kΩ BK5_TO_PLC
R38 220kΩ BK6_TO_PLC
R39 220kΩ BK7_TO_PLC
R40 220kΩ BK8_TO_PLC
R41 220kΩ BK9_TO_PLC
R42 220kΩ BK10_TO_PLC
R43 220kΩ CAL_CMP_TO_PLC
R44 220kΩ LIMIT_TO_PLC

8.8 A8 Non-Intrusive Pipe Pressure Measuring Circuit Schematic



8.9 A9 9V Regulator Circuit Schematic



8.10 A10 Clutch and Brake Pad Wear Test Transmission Program

```
/* Clutch and Brake Wear Monitoring */

#include <SPI.h>
#include <nRF24L01.h>
#include <RF24.h>
RF24 radio(53, 48); // CE, CSN
const byte address[6] = "00001";
void setup() {
  radio.begin();
  radio.openWritingPipe(address);
  radio.setPALevel(RF24_PA_MIN);
  radio.stopListening();
}
void loop() {
  const char text[] = "1NO";
  radio.write(&text, sizeof(text));
  delay(1000);

}
```

8.11 A11 Non-Intrusive Pipe Pressure Test Transmission Program

```
/*Non-Intrusive Pipe Pressure Monitoring */

#include <SPI.h>
#include <nRF24L01.h>
#include <RF24.h>
RF24 radio(53, 48); // CE, CSN
const byte address[6] = "00100";
void setup() {
  radio.begin();
  radio.openWritingPipe(address);
  radio.setPALevel(RF24_PA_MIN);
  radio.stopListening();
}
void loop() {
  const char text[] = "306";
  radio.write(&text, sizeof(text));
  delay(1000);
}
```

8.12 A12 Vibration Analysis Serial Test Transmission Program

```python
import serial
import time

ser = serial.Serial('/dev/ttyACM0', 9600)
time.sleep(5)
ser.flushInput()

while True:
    # Send 1 for 2OK, 2 for 2WE, 3 for 2NO, 4 for 2LE and 5 for 2ER
    # Conversion in Arduino from ASCII char to Decimal Number
    # 1 = 49 in Ard, 2 = 50 in Ard, 3 = 51 in Ard, 4 = 52 in Ard, 5 = 53 in Ard
    ser.write(b'1')
    time.sleep(1)
```

8.13 A13 Vibration Analysis Arduino Test Transmission Program

```
/*Vibration Analysis Monitoring */

#include <SPI.h>
#include <nRF24L01.h>
#include <RF24.h>
RF24 radio(53, 48); // CE, CSN
const byte address[6] = "00010";
int r = 0;
void setup() {
  radio.begin();
  radio.openWritingPipe(address);
  radio.setPALevel(RF24_PA_MIN);
  radio.stopListening();
  Serial.begin(9600);
}

void loop() {

  if(Serial.available()){
    r = Serial.read();
    //Serial.println(r);

      }
 if ( r == 49){       // 49 is the decimal equivalent of the ASCII char 1 which the RPi sends
      const char text1[] = "2OK";
      delay(100);
      radio.write(&text1, sizeof(text1));

    }
 if ( r == 50){       // 50 is the decimal equivalent of the ASCII char 2 which the RPi sends
      const char text2[] = "2WE";
      delay(100);
      radio.write(&text2, sizeof(text2));

     }
 if ( r == 51){       // 51 is the decimal equivalent of the ASCII char 3 which the RPi sends
      const char text3[] = "2NO";
      delay(100);
      radio.write(&text3, sizeof(text3));

    }
 if ( r == 52){       // 52 is the decimal equivalent of the ASCII char 4 which the RPi sends
      const char text4[] = "2LE";
      delay(100);
      radio.write(&text4, sizeof(text4));

    }
 if ( r == 53){       // 53 is the decimal equivalent of the ASCII char 5 which the RPi sends
      const char text5[] = "2ER";
      delay(100);
      radio.write(&text5, sizeof(text5));

    }
  delay(1000);}
```

8.14 A14 Central Graphical Device Receiver Test Transmission Program

```
#include <SPI.h>
#include <nRF24L01.h>
#include <RF24.h>
RF24 radio(53, 48); // CE, CSN
const byte numSlaves = 3;
const byte address[numSlaves][6] = {
  {"00001"},
  {"00010"},
  {"00100"}
};
String recv_buff[4];

void setup() {
  Serial.begin(9600);
  radio.begin();
  radio.setPALevel(RF24_PA_MIN);
  radio.startListening();
}

void loop() {
  for (byte n = 0; n < numSlaves; n++){
     radio.openReadingPipe(0, address[n]);
  if (radio.available()) {
    const char text[32] = "";
    radio.read(&text, sizeof(text));
    recv_buff[n] = text;
    String recv_tot = recv_buff[0] + recv_buff[1] + recv_buff[2];
    Serial.println(recv_tot);
    delay(2000);
    }


  }

}
```

8.15 A15 Central Graphical Device GUI Test Transmission Program

```
from appJar import gui

def launch(win):
   app.showSubWindow(win)
import serial
while True:
   ser = serial.Serial('/dev/ttyACM0',9600)
   read_serial=ser.readline()
   read_serial=read_serial.decode("utf-8").strip()
   rec_buff = str(read_serial)
   print(rec_buff)
   chkbuff1 = rec_buff[0:1]
   chkbuff2 = rec_buff[3:4]
   chkbuff3 = rec_buff[6:7]
   firstthreechars = rec_buff[0:3]
   secondthreechars = rec_buff[3:6]
   thirdthreechars = rec_buff[6:9]
   print (firstthreechars)
   print (secondthreechars)
   print (thirdthreechars)
   if chkbuff1 == '1'  and chkbuff2 == '2' and chkbuff3 == '3':
      break
   if chkbuff1 == '1'  and chkbuff2 == '3' and chkbuff3 == '2':
      break
   if chkbuff1 == '2'  and chkbuff2 == '1' and chkbuff3 == '3':
      break
   if chkbuff1 == '2'  and chkbuff2 == '3' and chkbuff3 == '1':
      break
   if chkbuff1 == '3'  and chkbuff2 == '1' and chkbuff3 == '2':
      break
   if chkbuff1 == '3'  and chkbuff2 == '2' and chkbuff3 == '1':
      break

app = gui("Condition Based Monitoring", "800x470")
app.setOnTop(stay=True)

app.addLabel("title", "CBM - C&B, VA and NIPP", 0, 0, 3)
app.setLabelBg("title", "white")
app.setLabelRelief("title","raised")
app.getLabelWidget("title").config(font=("Sans Serif", "20","bold","underline"))

app.addLabel("C&B1", "C&B", 1, 0, 1, 1)
app.setLabelBg("C&B1", "lightgray")
app.setLabelRelief("C&B1","ridge")
app.getLabelWidget("C&B1").config(font=("Sans Serif", "12","bold"))

if firstthreechars == '1OK' or secondthreechars == '1OK' or thirdthreechars == '1OK':
      app.addLabel("CB", "OK", 2, 0, 1, 1)
      app.addIcon("CB","check", 3, 0)
      app.setLabelBg("CB", "green")
      app.setLabelFg("CB", "white")
if firstthreechars == '1WE' or secondthreechars == '1WE' or thirdthreechars == '1WE':
      app.addLabel("CB", "Wearing", 2, 0, 1, 1)
      app.addIcon("CB","alert", 3, 0)
      app.setLabelBg("CB", "orange")
      app.setLabelFg("CB", "white")
if firstthreechars == '1NO' or secondthreechars == '1NO' or thirdthreechars == '1NO':
      app.addLabel("CB", "NOK",  2, 0, 1, 1)
```

```
    app.addIcon("CB","close", 3, 0)
    app.setLabelBg("CB", "red")
    app.setLabelFg("CB", "white")
if firstthreechars == '1LE' or secondthreechars == '1LE' or thirdthreechars == '1LE':
    app.addLabel("CB", "Limit!",  2, 0, 1, 1)
    app.addIcon("CB","balance", 3, 0)
    app.setLabelBg("CB", "blue")
    app.setLabelFg("CB", "white")
if firstthreechars == '1ER' or secondthreechars == '1ER' or thirdthreechars == '1ER':
    app.addLabel("CB", "Error",2, 0, 1, 1)
    app.setLabelFg("CB","red")
    app.addIcon("CB","rss",3,0)

app.addLabel("VA1", "VA", 1, 1, 1, 1)
app.setLabelBg("VA1", "lightgray")
app.setLabelRelief("VA1","ridge")
app.getLabelWidget("VA1").config(font=("Sans Serif", "12","bold"))

if firstthreechars == '2OK' or secondthreechars == '2OK' or thirdthreechars == '2OK':
    app.addLabel("VA", "OK", 2, 1, 1, 1)
    app.addIcon("VA","check", 3, 1)
    app.setLabelBg("VA", "green")
    app.setLabelFg("VA", "white")
if firstthreechars == '2WE' or secondthreechars == '2WE' or thirdthreechars == '2WE':
    app.addLabel("VA", "Wearing", 2, 1, 1, 1)
    app.addIcon("VA","alert", 3, 1)
    app.setLabelBg("VA", "orange")
    app.setLabelFg("VA", "white")
if firstthreechars == '2NO' or secondthreechars == '2NO' or thirdthreechars == '2NO':
    app.addLabel("VA", "NOK",  2, 1, 1, 1)
    app.addIcon("VA","close", 3, 1)
    app.setLabelBg("VA", "red")
    app.setLabelFg("VA", "white")
if firstthreechars == '2LE' or secondthreechars == '2LE' or thirdthreechars == '2LE':
    app.addLabel("VA", "Limit!",  2, 1, 1, 1)
    app.addIcon("VA","balance", 3, 1)
    app.setLabelBg("VA", "blue")
    app.setLabelFg("VA", "white")
if firstthreechars == '2ER' or secondthreechars == '2ER' or thirdthreechars == '2ER':
    app.addLabel("VA","Error", 2, 1, 1, 1)
    app.setLabelFg("VA","red")
    app.addIcon("VA","rss", 3, 1)

app.addLabel("NIPP1", "NIPP", 1, 2, 1, 1)
app.setLabelBg("NIPP1", "lightgray")
app.setLabelRelief("NIPP1","ridge")
app.getLabelWidget("NIPP1").config(font=("Sans Serif", "12","bold"))

if firstthreechars == '301' or secondthreechars == '301' or thirdthreechars == '301':
    app.addLabel("NI", "0-10 bar", 2, 2, 1, 1)
    app.addIcon("NI","science", 3, 2)
    app.setLabelBg("NI", "lightblue")
    app.setLabelFg("NI", "white")
if firstthreechars == '302' or secondthreechars == '302' or thirdthreechars == '302':
    app.addLabel("NI", "10-20 bar", 2, 2, 1, 1)
    app.addIcon("NI","science", 3, 2)
    app.setLabelBg("NI", "lightblue")
    app.setLabelFg("NI", "white")
if firstthreechars == '303' or secondthreechars == '303' or thirdthreechars == '303':
    app.addLabel("NI", "20-30 bar", 2, 2, 1, 1)
```

```
    app.addIcon("NI","science", 3, 2)
    app.setLabelBg("NI", "lightblue")
    app.setLabelFg("NI", "white")
if firstthreechars == '304' or secondthreechars == '304' or thirdthreechars == '304':
    app.addLabel("NI", "30-40 bar", 2, 2, 1, 1)
    app.addIcon("NI","science", 3, 2)
    app.setLabelBg("NI", "lightblue")
    app.setLabelFg("NI", "white")
if firstthreechars == '305' or secondthreechars == '305' or thirdthreechars == '305':
    app.addLabel("NI", "40-50 bar", 2, 2, 1, 1)
    app.addIcon("NI","science", 3, 2)
    app.setLabelBg("NI", "lightblue")
    app.setLabelFg("NI", "white")
if firstthreechars == '306' or secondthreechars == '306' or thirdthreechars == '306':
    app.addLabel("NI", "50-60 bar", 2, 2, 1, 1)
    app.addIcon("NI","science", 3, 2)
    app.setLabelBg("NI", "lightblue")
    app.setLabelFg("NI", "white")
if firstthreechars == '307' or secondthreechars == '307' or thirdthreechars == '307':
    app.addLabel("NI", "60-70 bar", 2, 2, 1, 1)
    app.addIcon("NI","science", 3, 2)
    app.setLabelBg("NI", "lightblue")
    app.setLabelFg("NI", "white")
if firstthreechars == '308' or secondthreechars == '308' or thirdthreechars == '308':
    app.addLabel("NI", "70-80 bar", 2, 2, 1, 1)
    app.addIcon("NI","science", 3, 2)
    app.setLabelBg("NI", "lightblue")
    app.setLabelFg("NI", "white")
if firstthreechars == '309' or secondthreechars == '309' or thirdthreechars == '309':
    app.addLabel("NI", "80-90 bar", 2, 2, 1, 1)
    app.addIcon("NI","science", 3, 2)
    app.setLabelBg("NI", "lightblue")
    app.setLabelFg("NI", "white")
if firstthreechars == '310' or secondthreechars == '310' or thirdthreechars == '310':
    app.addLabel("NI", "90-100 bar", 2, 2, 1, 1)
    app.addIcon("NI","science", 3, 2)
    app.setLabelBg("NI", "lightblue")
    app.setLabelFg("NI", "white")
if firstthreechars == '3LE' or secondthreechars == '3LE' or thirdthreechars == '3LE':
    app.addLabel("NI", "Limit!",  2, 2, 1, 1)
    app.addIcon("NI","balance", 3, 2)
    app.setLabelBg("NI", "blue")
    app.setLabelFg("NI", "white")
if firstthreechars == '3ER' or secondthreechars == '3ER' or thirdthreechars == '3ER':
    app.addLabel("NI", "Error",  2, 2, 1, 1)
    app.setLabelFg("NI", "red")
    app.addIcon("NI","rss", 3, 2)

app.addLabel("info","Exit this app and restart it to update", 4, 0 ,3)
app.setLabelFg("info","blue")
app.addLabel("info2","Press F5 to refresh once app is shutdown", 5, 0, 3)
app.setLabelFg("info2", "blue")
app.addButton("Exit",app.stop, 6,0,3)
app.setButtonFg("Exit","red")
app.go()
```

8.16 A16 Siemens PLC Integration STL Program

```
Block: FC600   Automatic CBM

Network: 1      Life Bit OUT to all MCs

      A     "ON"             M0.4
      =     "Life Bit OUT"  Q30.0              -- Life Bit to MC

Network: 2      Life Bit IN from  C&B MC

      A     "Life Bit IN C&B"    I30.0              -- Life Bit from MC
      =     "Life Bit Hold C&B"  M380.0             -- Life Bit Hold

Network: 3      Life Bit IN from  VA MC

      A     "Life Bit IN VA"    I40.0              -- Life Bit from MC
      =     "Life Bit Hold VA"  M480.0             -- Life Bit Hold

Network: 4      Life Bit IN from  NIPP MC

      A     "Life Bit IN NIPP"    I50.0              -- Life Bit from MC
      =     "Life Bit Hold NIPP"  M580.0             -- Life Bit Hold

Network: 5      Check Life Bit from C&B MC

      A     "Life Bit Hold C&B"   M380.0             -- Life Bit Hold
      L     S5T#2S
      SD    "Check Life Bit C&B"  T180               -- Check Life Bit from C&B MC
      NOP   0
      NOP   0
      NOP   0
      A     "Check Life Bit C&B"  T180               -- Check Life Bit from C&B MC
      =     "MC OK C&B"           M380.1             -- C&B MC OK from Life Bit
```

```
Network: 6      Check Life Bit from VA MC
```

```
A     "Life Bit Hold VA"   M480.0              -- Life Bit Hold
L     S5T#2S
SD    "Check Life Bit VA"  T181                -- Check Life Bit from VA MC
NOP   0
NOP   0
NOP   0
A     "Check Life Bit VA"  T181                -- Check Life Bit from VA MC
=     "MC OK VA"           M480.1              -- VA MC OK from Life Bit
```

```
Network: 7      Check Life Bit from NIPP MC
```

```
A     "Life Bit Hold NIPP"   M580.0              -- Life Bit Hold
L     S5T#2S
SD    "Check Life Bit NIPP"  T182                -- Check Life Bit from NIPP MC
NOP   0
NOP   0
NOP   0
A     "Check Life Bit NIPP"  T182                -- Check Life Bit from NIPP MC
=     "MC OK NIPP"           M580.1              -- NIPP MC OK from Life Bit
```

```
Network: 8      Enable Auto CBM
```

```
A     "Master ON"    M390.0              -- Master ON to enable AUTO CBM
A(
O     "MC OK C&B"    M380.1              -- C&B MC OK from Life Bit
O     "MC OK VA"     M480.1              -- VA MC OK from Life Bit
O     "MC OK NIPP"   M580.1              -- NIPP MC OK from Life Bit
)
=     "CBM Enabled"  M390.1              -- AUTO CBM Enabled
```

```
Network: 9      Enable C&B Auto CBM
```

```
A     "CBM Enabled"     M390.1              -- AUTO CBM Enabled
A     "Enable C&B CBM"  M390.4              -- Enable C&B CBM_
=     "C&B CBM"         M390.2              -- C&B CBM Enabled
```

```
Network: 10     Enable VA Auto CBM
```

```
A     "CBM Enabled"    M390.1              -- AUTO CBM Enabled
A     "Enable VA CBM"  M390.5              -- Enable VA CBM_
=     "VA CBM"         M390.3              -- VA CBM Enabled
```

```
Network: 11     Enable NIPP Auto CBM
```

```
A     "CBM Enabled"      M390.1              -- AUTO CBM Enabled
A     "Enable NIPP CBM"  M390.6              -- Enable NIPP CBM
=     "NIPP CBM"         M390.7              --  NIPP CBM Enabled
```

```
Network: 12     C&B Section
```

```
Network: 13     Press at TDC signal to MC
```

```
A     "C&B CBM"           M390.2              -- C&B CBM Enabled
A     "TDC Cam Internal"  M348.0
=     "PRESS AT TDC"      Q10.0               -- PRESS AT TDC TO MC
```

Network: 14 Press Motive signal to MC
When Q0.1 is ON then Press is in Motion

```
A     "C&B CBM"               M390.2          -- C&B CBM Enabled
A     "Cam Clutch Solenoid"   Q0.1            -- Cam Clutch Solenoid via safety relay
=     "PRESS MOTIVE"          Q10.1           -- PRESS MOTIVE TO MC
```

Network: 15 Calibration complete

```
A     "C&B CBM"               M390.2          -- C&B CBM Enabled
A     "Calibration Comp C&B"  I30.1           -- Calibration Complete C&B
=     "Calibration Comp"      M393.0          -- Calibration Complete C&B
```

Network: 16 Map microcontroller outputs to PLC inputs to memory bits

```
A     "C&B CBM"               M390.2          -- C&B CBM Enabled
A     "Calibration Comp"      M393.0          -- Calibration Complete C&B
=     L      0.0
A     L      0.0
A     "C&B OK"                I30.2           -- C&B CHECKED OK
=     "C&B WINDOW OK"         M396.0          -- C&B CHECK OK WINDOW
A     L      0.0
A     "C&B WE"                I30.3           -- C&B CHECKED WEARING
=     "C&B WINDOW WEARING"    M396.1          -- C&B CHECK WEARING WINDOW
A     L      0.0
A     "C&B NOK"               I30.4           -- C&B CHECKED NOK
=     "C&B WINDOW NOK"        M396.2          -- C&B CHECK NOK WINDOW
A     L      0.0
A     "C&B LE"                I30.5           -- C&B CHECKED LIMIT EXCEEDED
=     "Limited Exceeded Map"  M393.1          -- Limited Exceeded from MC C&B
```

Network: 17 C&B CBM Alarms

```
A     "C&B CBM"               M390.2          -- C&B CBM Enabled
=     L      0.0
A     L      0.0
A     "C&B WINDOW OK"         M396.0          -- C&B CHECK OK WINDOW
=     "C&B OK TO WINCC"       M602.0          -- C&B OK TO WINCC
A     L      0.0
A     "C&B WINDOW WEARING"    M396.1          -- C&B CHECK WEARING WINDOW
=     "C&B WEARING TO WINCC"  M602.1          -- C&B WEARING TO WINCC
A     L      0.0
A     "C&B WINDOW NOK"        M396.2          -- C&B CHECK NOK WINDOW
=     "C&B NOK TO WINCC"      M602.2          -- C&B NOK TO WINCC
A     L      0.0
A     "Limited Exceeded Map"  M393.1          -- Limited Exceeded from MC C&B
=     "C&B LIMIT EXCEED"      M602.3          -- C&B LIMIT EXCEEDED TO WINCC
```

Network: 18 C&B CBM Corrective Action 1

```
A     "C&B CBM"                                        M390.2        -- C&B CBM Enabled
A(
O     "C&B WINDOW WEARING"                             M396.1        -- C&B CHECK WEARING WINDOW
O     "C&B WINDOW NOK"                                 M396.2        -- C&B CHECK NOK WINDOW
)
A(
L     "C2 STORE"                                       MW1394        -- C2 STORE
L     0
==I
)
A(
L     "Current Job Data".Job_Data.Ram_Brake_Position   DB1.DBW42
L     300
>I
)
JNB   _001
L     "Current Job Data".Job_Data.Ram_Brake_Position   DB1.DBW42
L     2
-I
T     "Current Job Data".Job_Data.Ram_Brake_Position   DB1.DBW42
AN    OV
SAVE
```

```
      CLR
_001: A      BR
      =      L      0.0
      A      L      0.0
      BLD    102
      CU     "Count C&B Actions"                          C2                  -- Count C&B  Corrective Actions
      A      L      0.0
      JNB    _002
      L      "Count C&B Actions"                          C2                  -- Count C&B  Corrective Actions
      T      "C2 STORE"                                   MW1394              -- C2 STORE
_002: NOP    0
```

```
Network: 19     C&B CBM Corrective Action 2
```

```
      A      "C&B CBM"             M390.2            -- C&B CBM Enabled
      A(
      O      "C&B WINDOW WEARING"  M396.1            -- C&B CHECK WEARING WINDOW
      O      "C&B WINDOW NOK"      M396.2            -- C&B CHECK NOK WINDOW
      )
      A(
      L      "C2 STORE"            MW1394            -- C2 STORE
      L      1
      ==I
      )
      L      S5T#15M
      SD     "PAUSE AFTER ACTION"  T193              -- 15M PAUSE AFTER FIRST CORRECTIVE ACTION
      NOP    0
      NOP    0
      NOP    0
      A      "PAUSE AFTER ACTION"  T193              -- 15M PAUSE AFTER FIRST CORRECTIVE ACTION
      =      L      0.0
      A      L      0.0
      A(
      L      "RAM LOWER WINDOW"    MW1380            -- RAM LOWER WINDOW LIMIT
      L      340
      >I
      )
      JNB    _003
      L      "RAM LOWER WINDOW"    MW1380            -- RAM LOWER WINDOW LIMIT
      L      5
      -I
      T      "RAM LOWER WINDOW"    MW1380            -- RAM LOWER WINDOW LIMIT
_003: NOP    0
      A      L      0.0
      A(
      L      "RAM UPPER WINDOW"    MW1382            -- RAM UPPER WINDOW LIMIT
      L      20
      <I
      )
      JNB    _004
      L      "RAM UPPER WINDOW"    MW1382            -- RAM UPPER WINDOW LIMIT
      L      5
      +I
      T      "RAM UPPER WINDOW"    MW1382            -- RAM UPPER WINDOW LIMIT
      AN     OV
      SAVE
      CLR
_004: A      BR
      =      L      0.1
      A      L      0.1
      BLD    102
      CU     "Count C&B Actions"   C2                -- Count C&B  Corrective Actions
      A      L      0.1
      JNB    _005
      L      "Count C&B Actions"   C2                -- Count C&B  Corrective Actions
      T      "C2 STORE"            MW1394            -- C2 STORE
_005: NOP    0
```

Network: 20   C&B CBM Corrective Action Complete

```
      A     "C&B CBM"               M390.2           -- C&B CBM Enabled
      A(
      L     "C2 STORE"              MW1394           -- C2 STORE
      L     2
      ==I
      )
      =     L      0.0
      A     L      0.0
      BLD   102

      =     "ALL CORR ACT COMPLETE"  M601.3          -- ALL CORRECTIVE ACTIONS COMPLETED
      A     L      0.0
      JNB   _006
      L     0
      T     "C2 STORE"              MW1394           -- C2 STORE
      SET
      SAVE
      CLR
_006: A     BR
      L     MW  2874
      S     "Count C&B Actions"     C2               -- Count C&B  Corrective Actions
```

Network: 21   VA Section

Network: 22   Press at bottom signal

```
      A     "VA CBM"                M390.3           -- VA CBM Enabled
      A(
      L     "Main Ram Position Deg"  MW32
      L     170
      >=I
      )
      A(
      L     "Main Ram Position Deg"  MW32
      L     190
      <I
      )
      =     "Press at botttom"      Q10.2            -- Press at bottom VA
```

Network: 23   VA CBM

```
      A     "VA CBM"           M390.3           -- VA CBM Enabled
      =     L      0.0
      A     L      0.0
      A     "VA OK"            I20.0            -- VA OK from MC
      =     "VA OK Bit"        M600.0           -- VA OK Bit_
      A     L      0.0
      A     "VA Wearing"       I20.1            -- VA Wearing from MC
      =     "VA Wearing Bit_"  M600.1           -- VA Wearing Bit_
      A     L      0.0
      A     "VA NOK"           I20.2            -- VA NOK from MC
      =     "VA NOK Bit"       M600.2           -- VA NOK Bit
      A     L      0.0
      A     "VA LE"            I20.3            -- VA Limit Exceeded from MC
      =     "VA LE Bit"        M600.3           -- VA LE Bit_
```

Network: 24   VA CBM Alarms

```
      A     "VA CBM"            M390.3           -- VA CBM Enabled
      =     L      0.0
      A     L      0.0
      A     "VA OK Bit"         M600.0           -- VA OK Bit_
      =     "VA OK Alarm"       M601.0           -- VA OK Alarm
      A     L      0.0
      A     "VA Wearing Bit_"   M600.1           -- VA Wearing Bit_
      =     "VA Wearing Alarm"  M601.1           -- VA Wearing Alarm
      A     L      0.0
      A     "VA NOK Bit"        M600.2           -- VA NOK Bit
      =     "VA NOK Alarm"      M601.2           -- VA NOK Alarm
      A     L      0.0
      A     "VA LE Bit"         M600.3           -- VA LE Bit_
      =     "VA LE Alarm"       M601.4           -- VA LE Alarm
```

```
Network: 25     NIPP Section
```

```
Network: 26     Bracketone Mapping

      A     "NIPP CBM"          M390.7            --  NIPP CBM Enabled
      A     "BR 1 NIPP"         I20.4             -- Bracketone NIPP Signal
      =     L      0.0
      A     L      0.0
      BLD   102
      =     "BR 1 NIPP MAPPED"  M750.0            -- Bracketone NIPP Signal Mapped
      A     L      0.0
      JNB   _007
      L     10
      T     "Pressure NIPP"     MW4106            -- Upper Bound of Current Pressure
_007: NOP   0
```

```
Network: 27     Brackettwo Mapping

      A     "NIPP CBM"          M390.7            --  NIPP CBM Enabled
      A     "BR 2 NIPP"         I20.5             -- Brackettwo NIPP Signal
      =     L      0.0
      A     L      0.0
      BLD   102
      =     "BR 2 NIPP MAPPED"  M750.1            -- Brackettwo NIPP Signal Mapped
      A     L      0.0
      JNB   _008
      L     20
      T     "Pressure NIPP"     MW4106            -- Upper Bound of Current Pressure
_008: NOP   0
```

```
Network: 28     Bracketthree Mapping

      A     "NIPP CBM"          M390.7            --  NIPP CBM Enabled
      A     "BR 3 NIPP"         I20.6             -- Bracketthree NIPP Signal
      =     L      0.0
      A     L      0.0
      BLD   102
      =     "BR 3 NIPP MAPPED"  M750.2            -- Bracketthree NIPP Signal Mapped
      A     L      0.0
      JNB   _009
      L     30
      T     "Pressure NIPP"     MW4106            -- Upper Bound of Current Pressure
_009: NOP   0
```

```
Network: 29     Bracketfour Mapping

      A     "NIPP CBM"          M390.7            --  NIPP CBM Enabled
      A     "BR 4 NIPP"         I20.7             -- Bracketfour NIPP Signal
      =     L      0.0
      A     L      0.0
      BLD   102
      =     "BR 4 NIPP MAPPED"  M750.3            -- Bracketfour NIPP Signal Mapped
      A     L      0.0
      JNB   _00a
      L     40
      T     "Pressure NIPP"     MW4106            -- Upper Bound of Current Pressure
_00a: NOP   0
```

```
Network: 30     Bracketfive Mapping

      A     "NIPP CBM"          M390.7            --  NIPP CBM Enabled
      A     "BR 5 NIPP"         I22.0             -- Bracketfive NIPP Signal
      =     L      0.0
      A     L      0.0
      BLD   102
      =     "BR 5 NIPP MAPPED"  M750.4            -- Bracketfive NIPP Signal Mapped
      A     L      0.0


      JNB   _00b
      L     50
      T     "Pressure NIPP"     MW4106            -- Upper Bound of Current Pressure
_00b: NOP   0
```

```
Network: 31     Bracketsix Mapping

      A     "NIPP CBM"          M390.7              --  NIPP CBM Enabled
      A     "BR 6 NIPP"         I22.1               -- Bracketsix NIPP Signal
      =     L      0.0
      A     L      0.0
      BLD   102
      =     "BR 6 NIPP MAPPED"  M750.5              -- Bracketsix NIPP Signal Mapped
      A     L      0.0
      JNB   _00c
      L     60
      T     "Pressure NIPP"     MW4106              -- Upper Bound of Current Pressure
_00c: NOP   0
```

```
Network: 32     Bracketseven Mapping

      A     "NIPP CBM"          M390.7              --  NIPP CBM Enabled
      A     "BR 7 NIPP"         I22.2               -- Bracketseven NIPP Signal
      =     L      0.0
      A     L      0.0
      BLD   102
      =     "BR 7 NIPP MAPPED"  M750.6              -- Bracketseven NIPP Signal Mapped
      A     L      0.0
      JNB   _00d
      L     70
      T     "Pressure NIPP"     MW4106              -- Upper Bound of Current Pressure
_00d: NOP   0
```

```
Network: 33     Bracketeight Mapping

      A     "NIPP CBM"          M390.7              --  NIPP CBM Enabled
      A     "BR 8 NIPP"         I22.3               -- Bracketeight NIPP Signal
      =     L      0.0
      A     L      0.0
      BLD   102
      =     "BR 8 NIPP MAPPED"  M750.7              -- Bracketeight NIPP Signal Mapped
      A     L      0.0
      JNB   _00e
      L     80
      T     "Pressure NIPP"     MW4106              -- Upper Bound of Current Pressure
_00e: NOP   0
```

```
Network: 34     Bracketnine Mapping

      A     "NIPP CBM"          M390.7              --  NIPP CBM Enabled
      A     "BR 9 NIPP"         I22.4               -- Bracketnine NIPP Signal
      =     L      0.0
      A     L      0.0
      BLD   102
      =     "BR 9 NIPP MAPPED"  M751.0              -- Bracketnine NIPP Signal Mapped
      A     L      0.0
      JNB   _00f
      L     90
      T     "Pressure NIPP"     MW4106              -- Upper Bound of Current Pressure
_00f: NOP   0
```

```
Network: 35     Bracketten Mapping

      A     "NIPP CBM"           M390.7              --  NIPP CBM Enabled
      A     "BR 10 NIPP"         I22.5               -- Bracketten NIPP Signal
      =     L      0.0
      A     L      0.0
      BLD   102
      =     "BR 10 NIPP MAPPED"  M751.1              -- Bracketten NIPP Signal Mapped
      A     L      0.0
      JNB   _010
      L     100
      T     "Pressure NIPP"      MW4106              -- Upper Bound of Current Pressure

_010: NOP   0
```

Network: 36   Pumps On Output Mapping

```
A     "NIPP CBM"       M390.7            --  NIPP CBM Enabled
A     "Pumps are ON"   M3075.0           -- Pumps are ON NIPP
=     "PUMPS ON NIPP"  Q10.3             -- Pumps On Signal to NIPP MC
```

Network: 37   Calibration Complete Input Mapping

```
A     "NIPP CBM"         M390.7            --  NIPP CBM Enabled
A     "Calib Comp NIPP"  I22.6             -- Calibration Complete Signal from NIPP MC
=     "NIPP Calib Comp"  M3075.1           -- Calibration Complete NIPP
```

Network: 38   Limit Exceeded Input Mapping

```
A     "NIPP CBM"        M390.7            --  NIPP CBM Enabled
A     "NIPP LE"         I22.7             -- NIPP Limit Exceed Signal from MC
=     "NIPP LE Mapped"  M3075.2           -- NIPP LE Signal Mapped
```

Network: 39   NIPP CBM Alarms

```
A     "NIPP CBM"           M390.7            --  NIPP CBM Enabled
A     "NIPP Calib Comp"    M3075.1           -- Calibration Complete NIPP
=     L      0.0
A     L      0.0
A     "BR 1 NIPP MAPPED"   M750.0            -- Bracketone NIPP Signal Mapped
=     "BR 1 ALARM"         M760.0            -- BR 1 ALARM NIPP FOR WINCC
A     L      0.0
A     "BR 2 NIPP MAPPED"   M750.1            -- Brackettwo NIPP Signal Mapped
=     "BR 2 ALARM"         M760.1            -- BR 2 ALARM NIPP FOR WINCC
A     L      0.0
A     "BR 3 NIPP MAPPED"   M750.2            -- Bracketthree NIPP Signal Mapped
=     "BR 3 ALARM"         M760.2            --  BR 3 ALARM NIPP FOR WINCC
A     L      0.0
A     "BR 4 NIPP MAPPED"   M750.3            -- Bracketfour NIPP Signal Mapped
=     "BR 4 ALARM"         M760.3            -- BR 4 ALARM NIPP FOR WINCC
A     L      0.0
A     "BR 5 NIPP MAPPED"   M750.4            -- Bracketfive NIPP Signal Mapped
=     "BR 5 ALARM"         M760.4            -- BR 5 ALARM NIPP FOR WINCC
A     L      0.0
A     "BR 6 NIPP MAPPED"   M750.5            -- Bracketsix NIPP Signal Mapped
=     "BR 6 ALARM"         M760.5            -- BR 6 ALARM NIPP FOR WINCC
A     L      0.0
A     "BR 7 NIPP MAPPED"   M750.6            -- Bracketseven NIPP Signal Mapped
=     "BR 7 ALARM"         M760.6            -- BR 7 ALARM NIPP FOR WINCC
A     L      0.0
A     "BR 8 NIPP MAPPED"   M750.7            -- Bracketeight NIPP Signal Mapped
=     "BR 8 ALARM"         M760.7            -- BR 8 ALARM NIPP FOR WINCC
A     L      0.0
A     "BR 9 NIPP MAPPED"   M751.0            -- Bracketnine NIPP Signal Mapped
=     "BR 9 ALARM"         M761.0            -- BR 9 ALARM NIPP FOR WINCC
A     L      0.0
A     "BR 10 NIPP MAPPED"  M751.1            -- Bracketten NIPP Signal Mapped
=     "BR 10 ALARM"        M761.1            -- BR 10 ALARM NIPP FOR WINCC
A     L      0.0
A     "NIPP LE Mapped"     M3075.2           -- NIPP LE Signal Mapped
=     "NIPP LE ALARM"      M3175.2           -- NIPP LE ALARM FOR WINCC
```

Network: 40   NIPP Pump Shutoff 1

```
      A(
      A     "NIPP CBM"              M390.7            --  NIPP CBM Enabled
      A     "Pumps are ON"          M3075.0           -- Pumps are ON NIPP
      A(
      L     "NIPP Pressure Recent"  MW4108            -- Most recent Pressure Reading this Scan
      L     "Pressure NIPP"         MW4106            -- Upper Bound of Current Pressure
      <>I
      )
      JNB   _011
      L     "NIPP Pressure Recent"  MW4108            -- Most recent Pressure Reading this Scan
      L     "Pressure NIPP"         MW4106            -- Upper Bound of Current Pressure
      /I

      T     "ROC NIPP"              MW4110            -- Rate of Change of Pressure between Scans NIPP
      AN    OV
      SAVE
      CLR
_011: A     BR                                                                                    _01
      )
      A(
      L     "ROC NIPP"              MW4110            -- Rate of Change of Pressure between Scans NIPP
      L     5
      >=I
      )
      =     L      0.0
      A     L      0.0
      BLD   102
      CU    "P Drop Counter"        C3                -- Pressure Drop Counter
      A     L      0.0
      JNB   _012
      L     "P Drop Counter"        C3                -- Pressure Drop Counter
      T     "C3 STORE"              MW4112            -- C3 STORE
_012: NOP   0                                                                                     _01
```

Network: 41   NIPP Pump Shutoff Confirmation

```
      A     "NIPP CBM"                 M390.7            --  NIPP CBM Enabled
      A     "Pumps are ON"             M3075.0           -- Pumps are ON NIPP
      A(
      L     "C3 STORE"                 MW4112            -- C3 STORE
      L     1
      ==I
      )
      A     "BR 1 NIPP"                I20.4             -- Bracketone NIPP Signal
      L     S5T#2S
      SD    "NIPP Check Timer"         T185              -- NIPP Check Pressure Dropped for 5s
      NOP   0
      NOP   0
      NOP   0
      A     "NIPP Check Timer"         T185              -- NIPP Check Pressure Dropped for 5s
      =     L      0.0
      A     L      0.0
      BLD   102
      =     "Cycle Stop This Press"    M343.5
      A     L      0.0
      BLD   102
      =     "NIPP Flag"                M4120.0           -- NIPP Flag Cycle Stop
      A(
      A(
      A     L      0.0
      L     S5T#2S
      SD    "Cycle Stop Trigger Buff"  T187              -- Cycle Stop Trigger Buffer
      NOP   0
      NOP   0
      NOP   0
      A     "Cycle Stop Trigger Buff"  T187              -- Cycle Stop Trigger Buffer
      )
      JNB   _013
      L     0
      T     "C3 STORE"                 MW4112            -- C3 STORE
      SET
      SAVE
      CLR
_013: A     BR
      )
      JNB   _014
      L     "C3 STORE"                 MW4112            -- C3 STORE
      T     "C3 BUFFER"                MW4114            -- C3 BUFFER
_014: NOP   0
      A     L      0.0
      BLD   102
      L     "C3 BUFFER"                MW4114            -- C3 BUFFER
      S     "P Drop Counter"           C3                -- Pressure Drop Counter
```

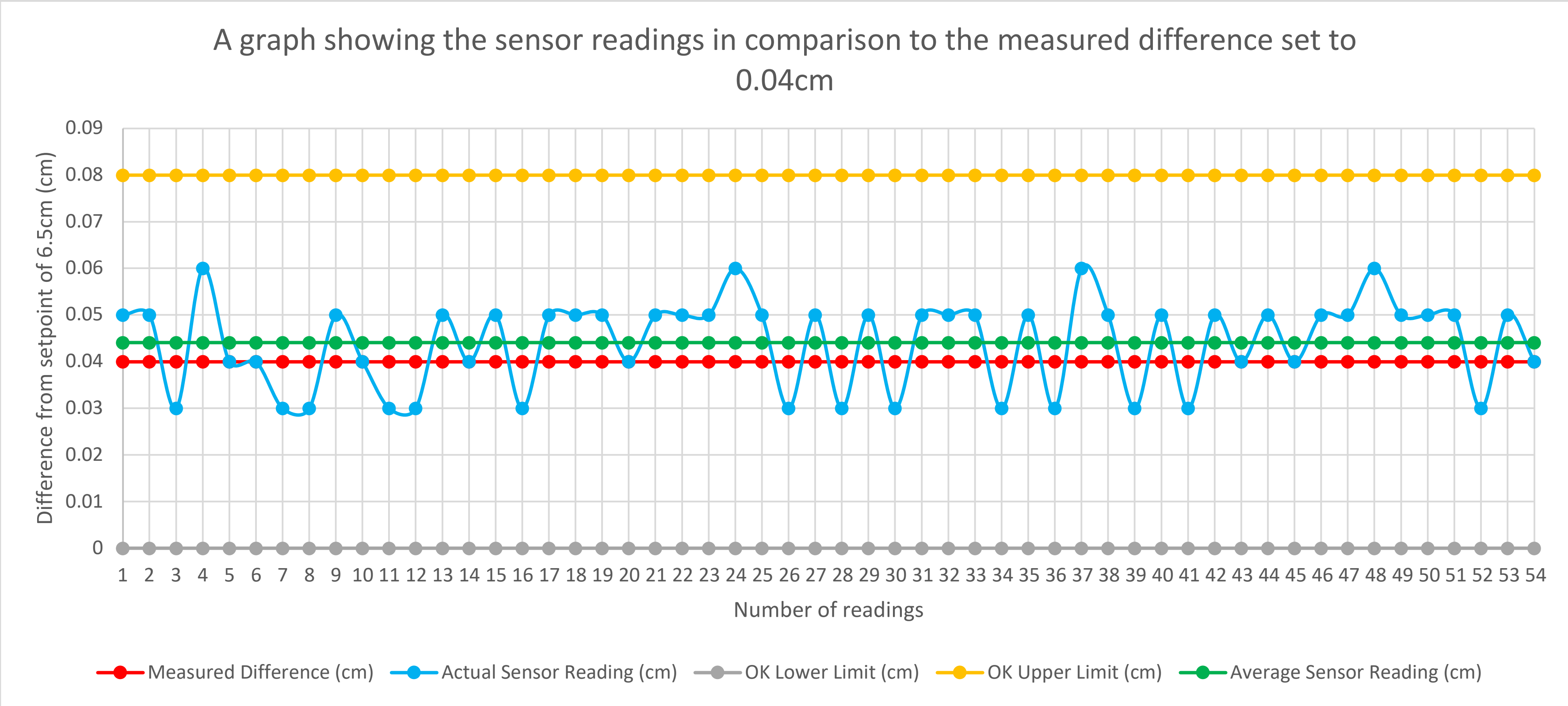
Network: 42   NIPP Pump Shutoff 2

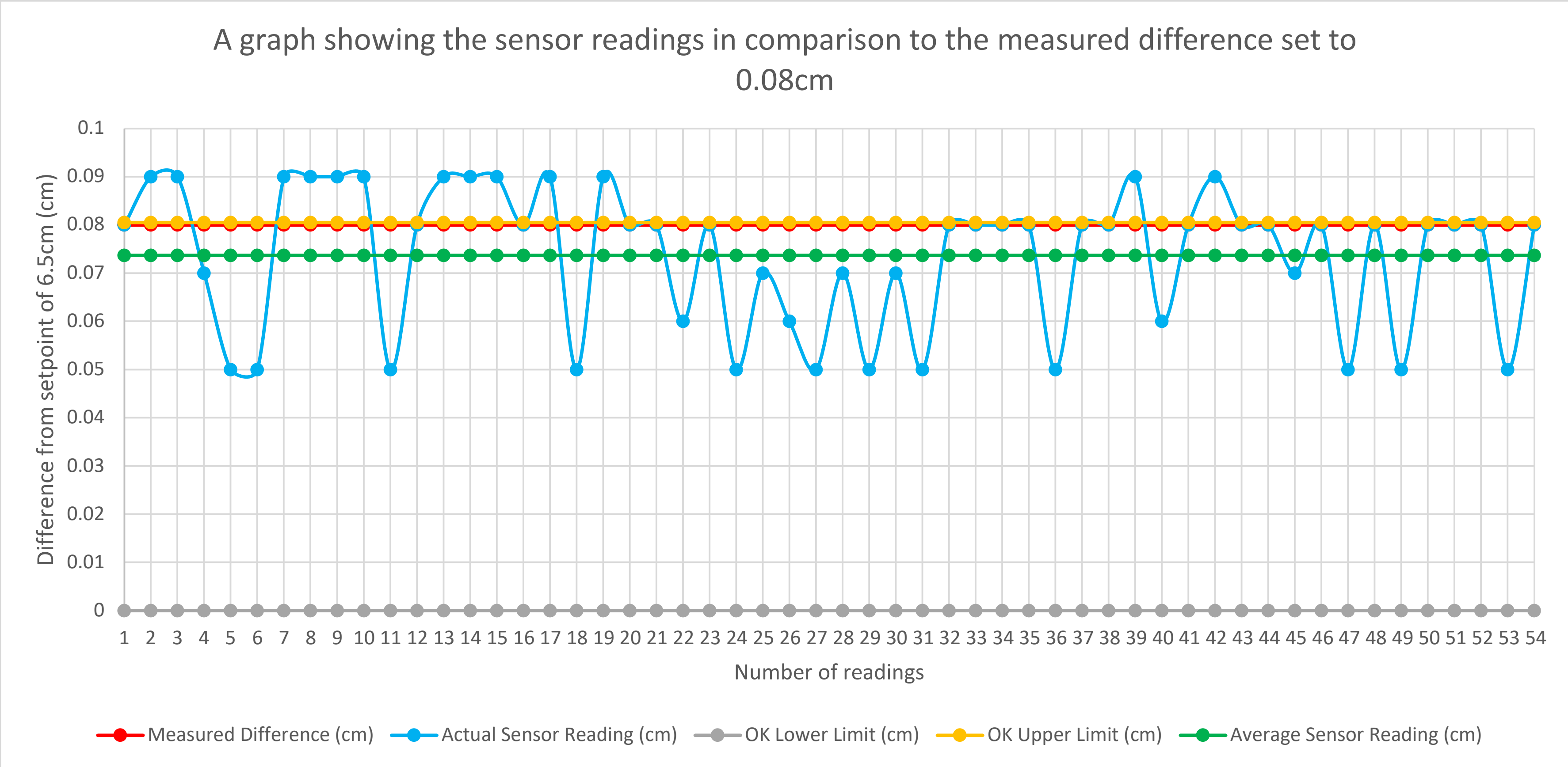
```
      A     "NIPP CBM"                 M390.7            --  NIPP CBM Enabled
      A     "Pumps are ON"             M3075.0           -- Pumps are ON NIPP
      A     "NIPP Flag"                M4120.0           -- NIPP Flag Cycle Stop
      A     "Cycle Stopped Confirmed"  M343.6
      L     S5T#500MS
      SD    "NIPP Press Stop Timer"    T186              -- NIPP Short Buffer Timer to Ensure Press has Cycle Stopped


      NOP   0
      NOP   0
      NOP   0
      A     "NIPP Press Stop Timer"    T186              -- NIPP Short Buffer Timer to Ensure Press has Cycle Stopped
      =     "NIPP SHUTDOWN ACTIVATED"  M3075.3           -- NIPP PUMP SHUTDOWN ACTIVATED ALARM
      =     "NIPP Pump Off"            M4120.1           -- NIPP Signal to turn Pumps off if safe
```

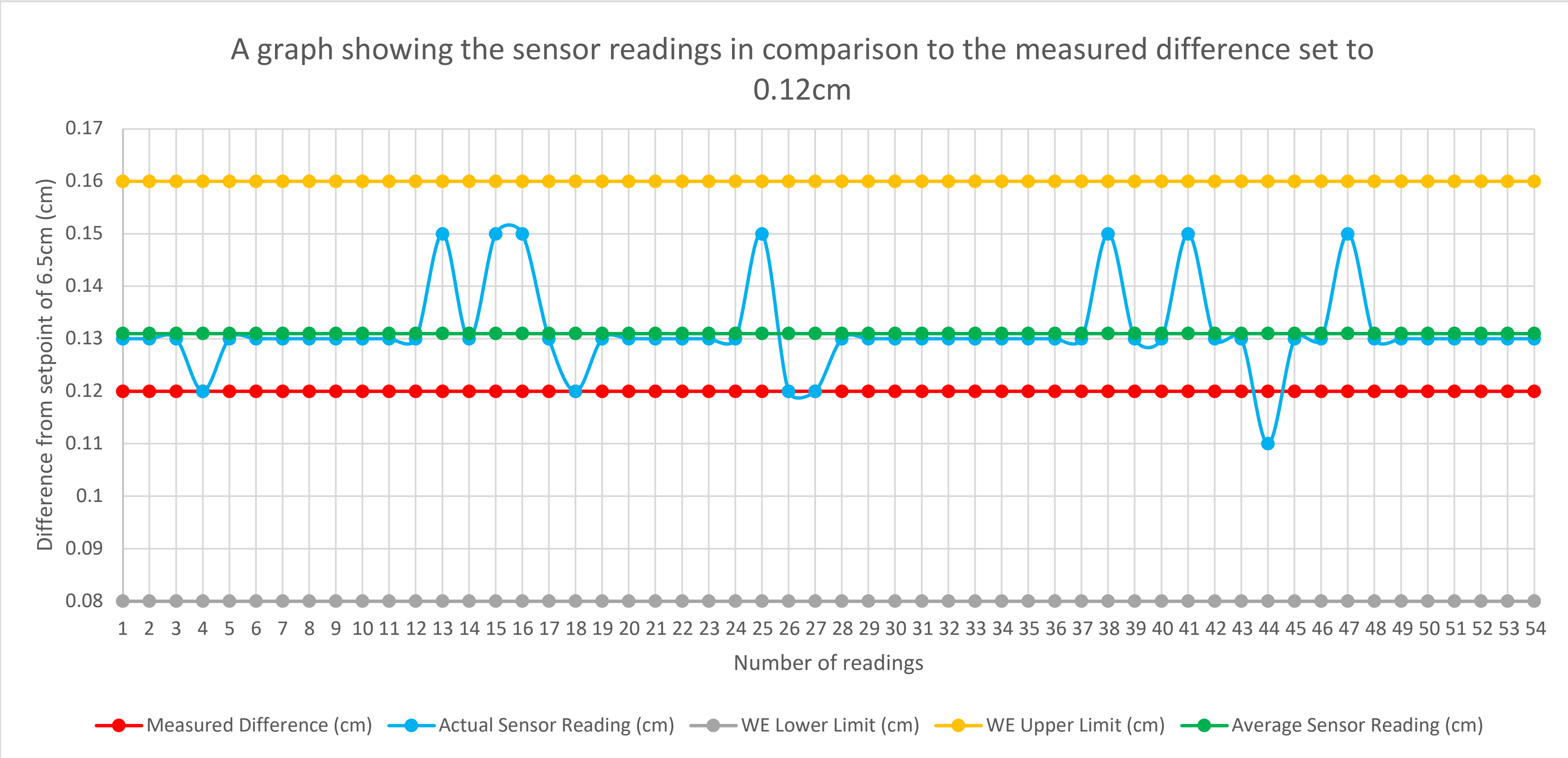
Network: 43   NIPP Pump Next Pressure Reading

```
      A     "NIPP CBM"               M390.7            --  NIPP CBM Enabled
      A     "Pumps are ON"           M3075.0           -- Pumps are ON NIPP
      JNB   _015
      L     "Pressure NIPP"          MW4106            -- Upper Bound of Current Pressure
      T     "NIPP Pressure Recent"   MW4108            -- Most recent Pressure Reading this Scan
_015: NOP   0
```

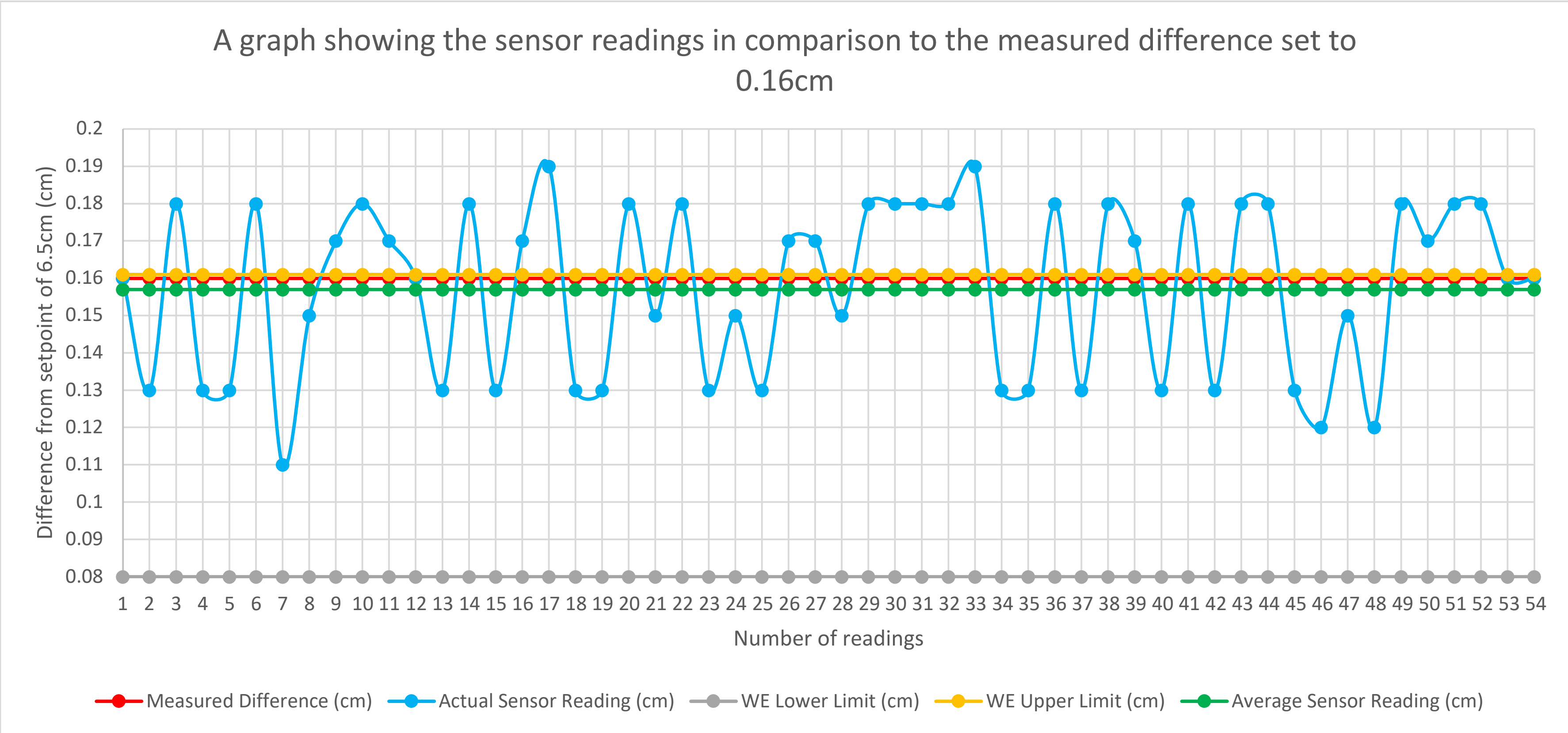
8.17 A17 Clutch and Brake Sensor Prototype Test Result Graphs



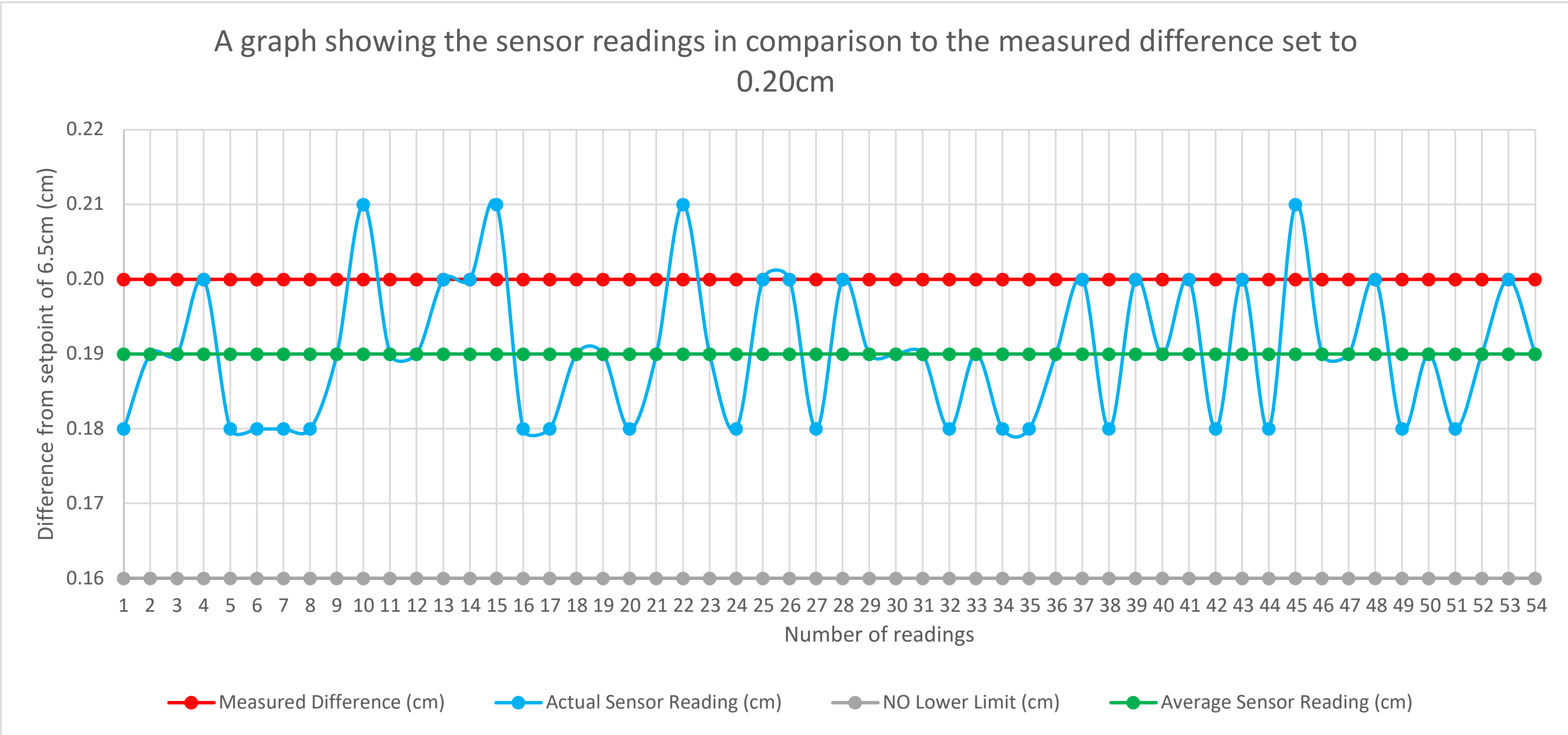

A graph showing the sensor readings in comparison to the measured difference set to 0.08cm
Difference from setpoint of 6.5cm (cm)
0.1
0.09
0.08
0.07
0.06
0.05
0.04
0.03
0.02
0.01
0
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54
Number of readings
Measured Difference (cm)
Actual Sensor Reading (cm)
OK Lower Limit (cm)
OK Upper Limit (cm)
Average Sensor Reading (cm)



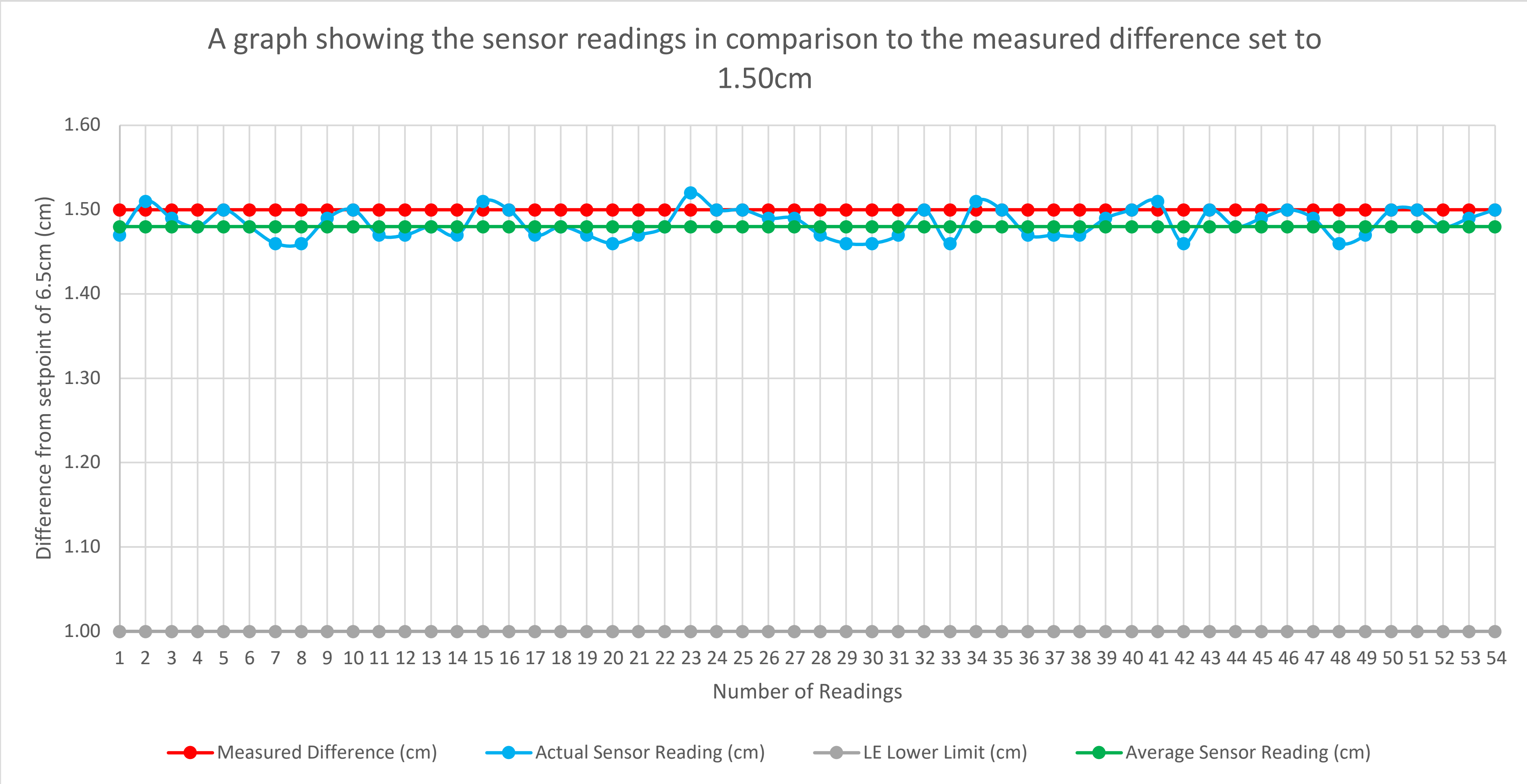
A graph showing the sensor readings in comparison to the measured difference set to 0.12cm
Difference from setpoint of 6.5cm (cm)
0.17
0.16
0.15
0.14
0.13
0.12
0.11
0.1
0.09
0.08
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54
Number of readings
Measured Difference (cm)
Actual Sensor Reading (cm)
WE Lower Limit (cm)
WE Upper Limit (cm)
Average Sensor Reading (cm)



A graph showing the sensor readings in comparison to the measured difference set to 0.16cm
Difference from setpoint of 6.5cm (cm)
0.2
0.19
0.18
0.17
0.16
0.15
0.14
0.13
0.12
0.11
0.1
0.09
0.08
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54
Number of readings
Measured Difference (cm)
Actual Sensor Reading (cm)
WE Lower Limit (cm)
WE Upper Limit (cm)
Average Sensor Reading (cm)

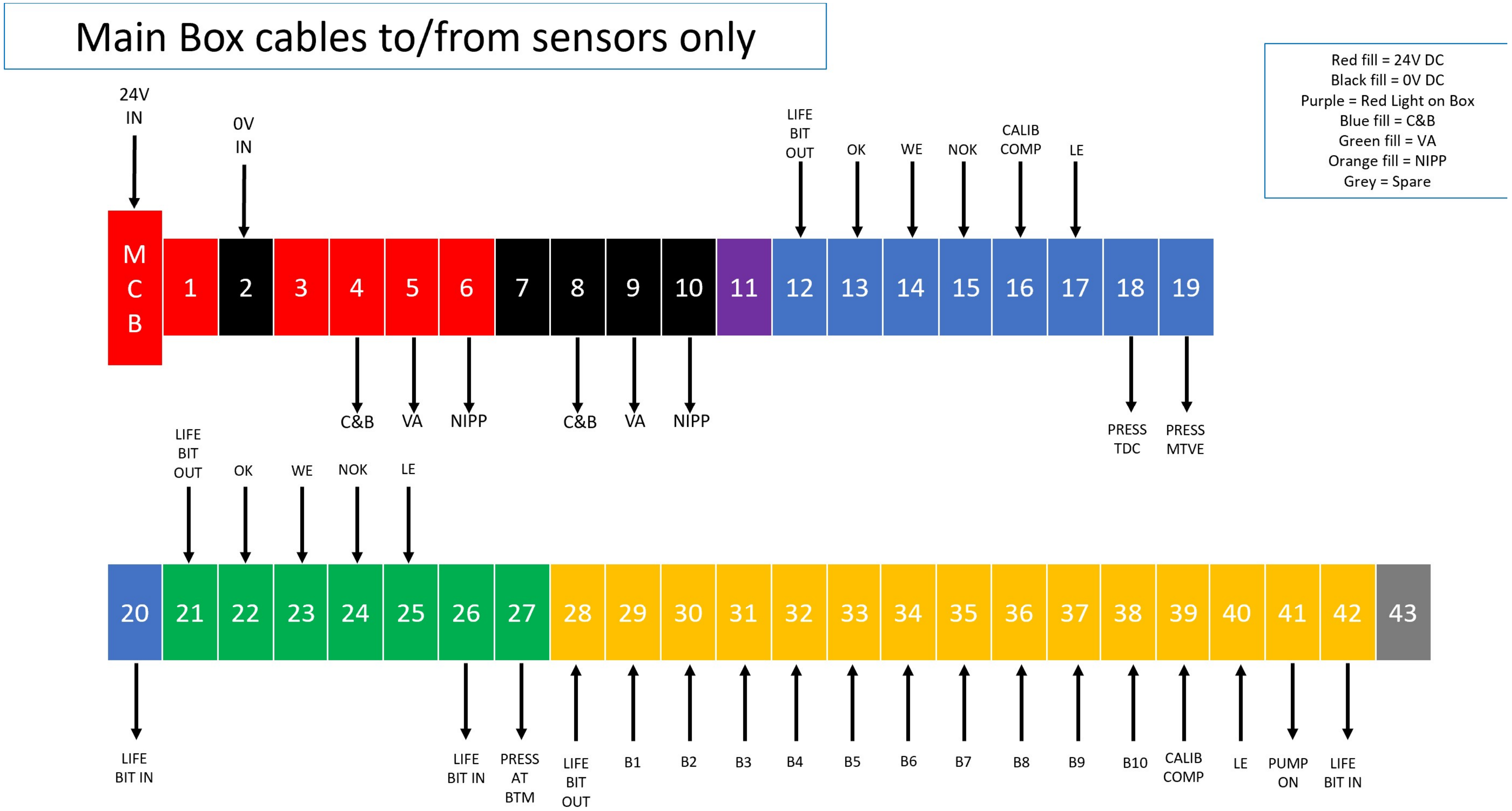


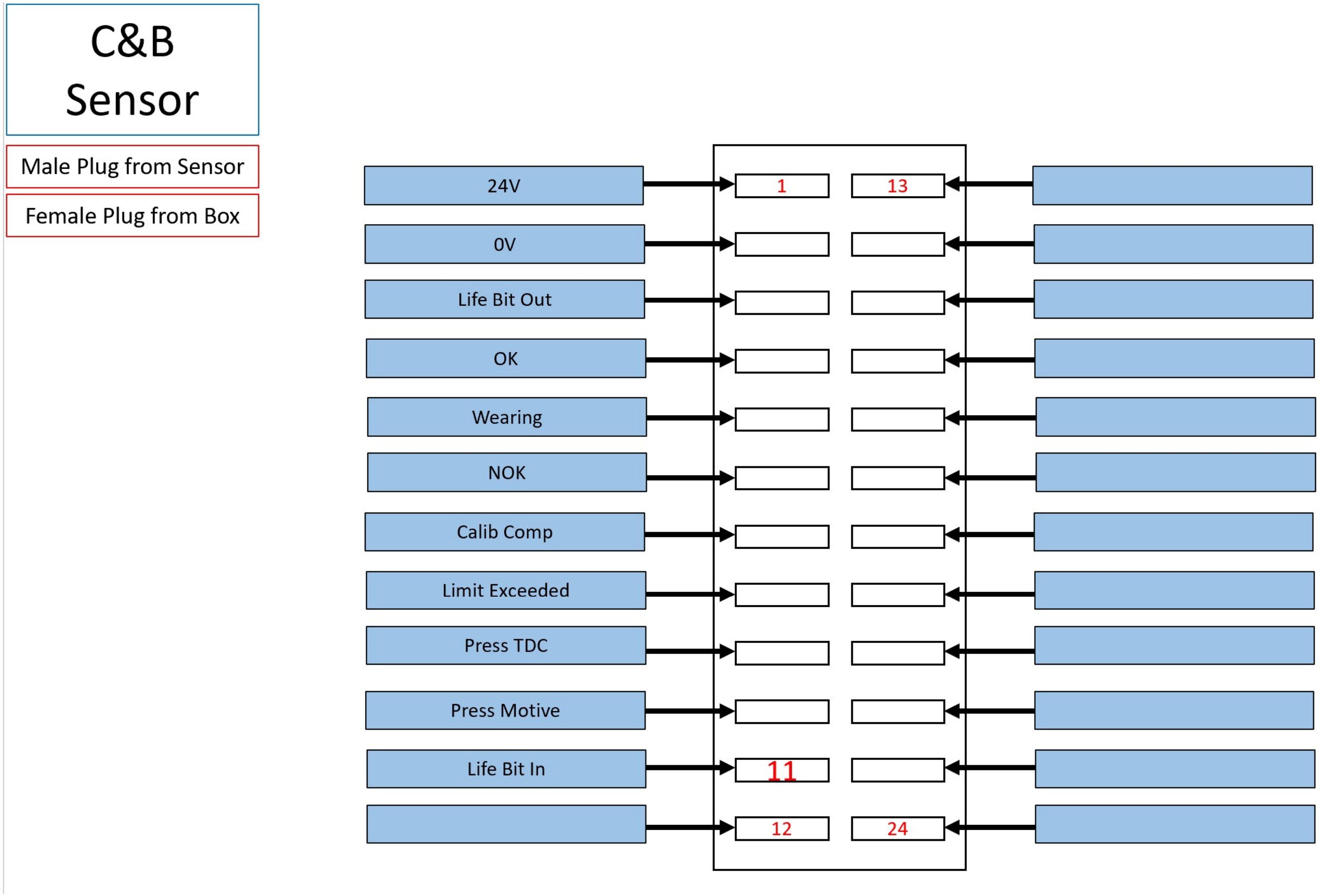
A graph showing the sensor readings in comparison to the measured difference set to 0.20cm
Difference from setpoint of 6.5cm (cm)
0.22
0.21
0.20
0.19
0.18
0.17
0.16
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54
Number of readings
Measured Difference (cm)
Actual Sensor Reading (cm)
NO Lower Limit (cm)
Average Sensor Reading (cm)



A graph showing the sensor readings in comparison to the measured difference set to 1.50cm
1.60
1.50
1.40
1.30
1.20
1.10
1.00
Difference from setpoint of 6.5cm (cm)
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54
Number of Readings
Measured Difference (cm)
Actual Sensor Reading (cm)
LE Lower Limit (cm)
Average Sensor Reading (cm)

8.18 A18 Connections Junction Box Schematics

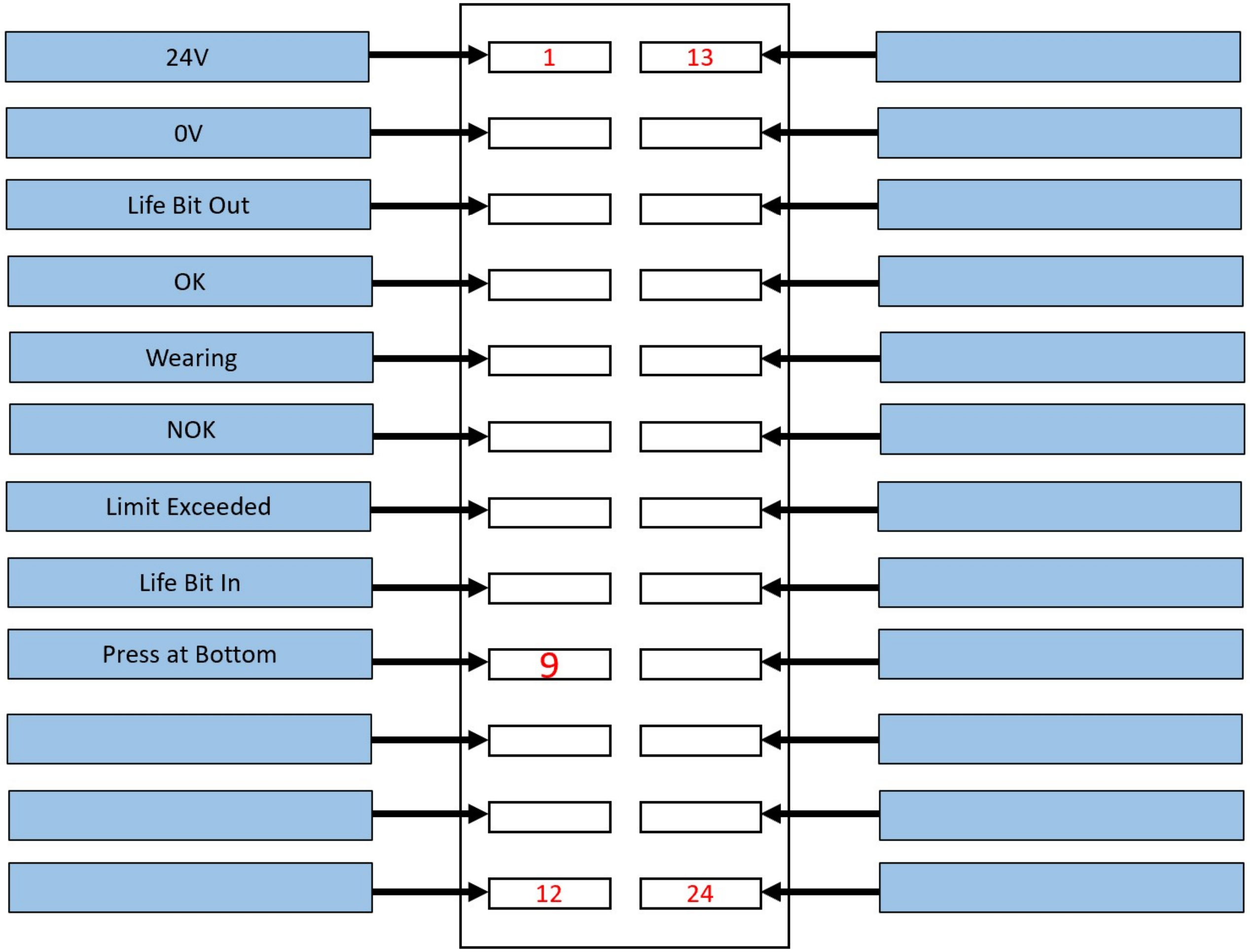



C&B
Sensor
Male Plug from Sensor
Female Plug from Box
24V
0V
Life Bit Out
OK
Wearing
NOK
Calib Comp
Limit Exceeded
Press TDC
Press Motive
Life Bit In
1
13
11
12
24

VA Sensor

Male Plug from Sensor

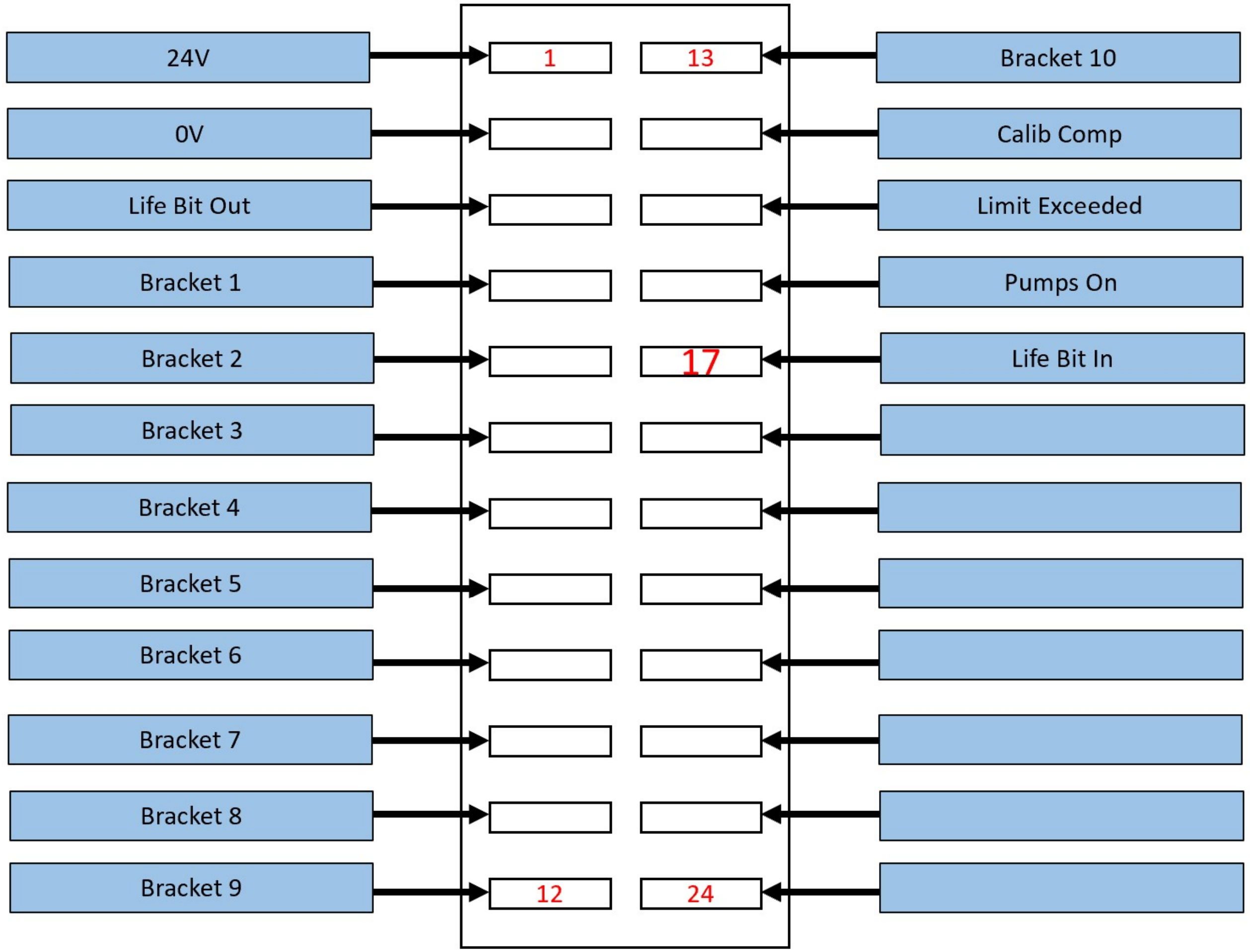
Female Plug from Box



NIPP Sensor

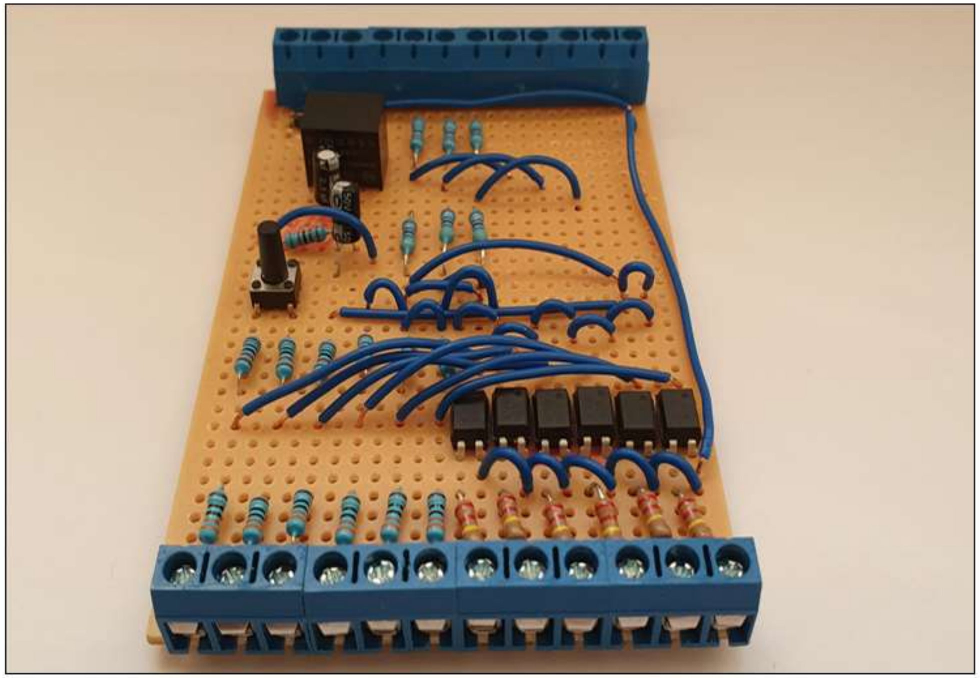
Male Plug from Sensor

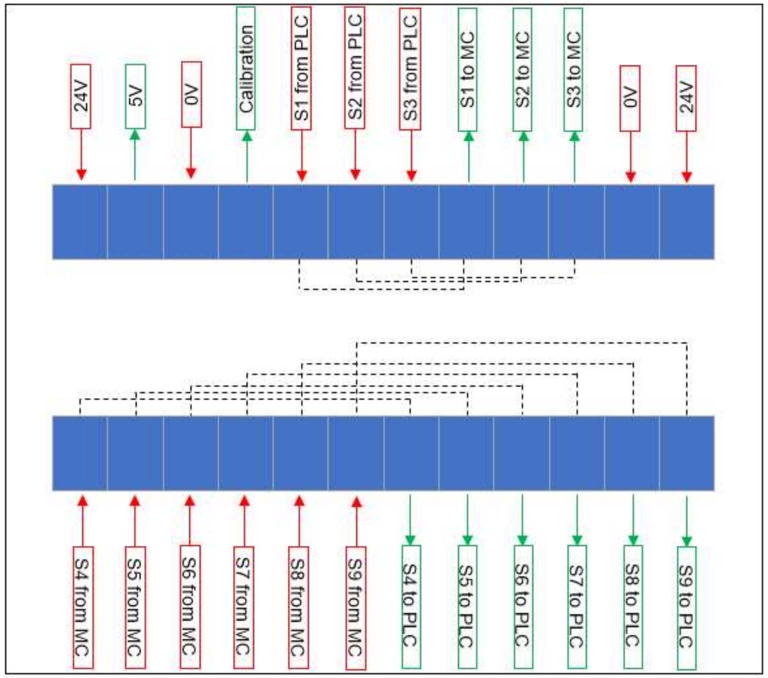
Female Plug from Box



8.19 A19 C&B Interfacing Circuit Photograph and Terminal Designations

C&B Interfacing and Calibration Circuit Board





Information: blue blocks represent the terminals on the board, a red arrow and box is an input to the board whereas green is an output from the board. Sx is a signal. Dashed lines connect signals that are related.

8.20 A20 Complete C&B Arduino Program

```
/* Clutch and Brake Wear Monitoring */

/* nRF Libraries */

#include <SPI.h>                // Include the SPI library
#include <nRF24L01.h>           // nRF Library
#include <RF24.h>               // A further nRF24 library

/* nRF Setup */

RF24 radio(53, 48); // CE, CSN    // Declare pins for SPI communications
const byte address[6] = "00001";  // Setup nRF address for C&B sensor
const char text[] = "";

/* Input and Output Declarations */

#define sensor A0               // Analog input from the IR sensor

const int calib = 30;           // Calibration button input to set a datum - hold for 5 seconds
const int presstdc = 31;        // Press at TDC signal from the PLC
const int pressmotive = 33;     // Press Motive signal from the PLC
const int lifebitin = 34;       // Life Bit signal from the PLC
const int lifebitout = 35;      // Life Bit signal to the PLC
const int calibcomp = 36;       // Calibration complete to allow measurement
const int OK = 37;              // Current condition OK signal to the PLC
const int Wearing = 38;         // Current condition Wearing signal to the PLC
const int NOK = 39;             // Current condition NOK signal to the PLC
const int LimitExceeded = 40;   // The boundary limits have been exceeded signal to the PLC

/* Setup Local Variables */

float difference0 = 0.000;      // First hold area for difference measurement
float difference1 = 0.000;      // Second hold area for difference measurement
float difference2 = 0.000;      // Third hold area for difference measurement
float difference3 = 0.000;      // Fourth hold area for difference measurement
float difference4 = 0.000;      // Fifth hold area for difference measurement
float difference5 = 0.000;      // Sixth hold area for difference measurement
float difference6 = 0.000;      // Seventh hold area for difference measurement
float difference7 = 0.000;      // Eighth hold area for difference measurement
float difference8 = 0.000;      // Ninth hold area for difference measurement
float difference9 = 0.000;      // Tenth hold area for difference measurement
float checkreading;             // Check each reading before confirmation
float avgdifference;            // Average of all the averages
float calibdistance;            // The initial measured calibration distance
float avgone;                   // The first average from the first ten measurements
float avgtwo;                   // The second average from the second ten measurements
float avgthree;                 // The third average from the third ten measurements

int calibcompinternal;          // Internal bit for calibration completion checking
int avgdifferencefirstcomp;     // Checks the first avg difference has been taken


void setup() {

  pinMode(calib, INPUT);        // Set calib pin as an input
```

```
pinMode(presstdc, INPUT);            // Set presstdc pin as an input
pinMode(pressmotive, INPUT);          // Set pressmotive pin as an input
pinMode(lifebitin, INPUT);           // Set lifebitin pin as an input
pinMode(lifebitout, OUTPUT);          // Set lifebitout pin as an output
pinMode(OK, OUTPUT);                 // Set OK pin as an output
pinMode(Wearing, OUTPUT);             // Set Wearing pin as an output
pinMode(NOK, OUTPUT);                 // Set NOK pin as an output
pinMode(calibcomp, OUTPUT);            // Set calibcomp pin as an output
pinMode(LimitExceeded, OUTPUT);        // Set LimitExceed as an output
digitalWrite(calibcomp, LOW);         // Set calibcomp low on power cycle
radio.begin();                      // Initialise the nRF system
radio.openWritingPipe(address);       // Open the writing pipe to address 00001
radio.setPALevel(RF24_PA_MIN);         // Power Amplifier set to minimum level
radio.stopListening();               // nRF is told to stop listening so it can write
          }
void loop() {
 if (digitalRead(lifebitin) == HIGH){
// Check lifebit from PLC is OK
  digitalWrite(lifebitout, HIGH);
// Send lifebit to PLC
  if (digitalRead(presstdc) == HIGH & digitalRead(pressmotive) == LOW){
// Check for Press at TDC and Press Motive off
    if (digitalRead(calib) == HIGH){
 // Check calibration button is pressed
      float measurement = analogRead(sensor);
 // Read the value from the sensor
      float measasvolt = (measurement * 0.0048828125);
// Convert value from sensor to a voltage
      calibdistance =  ((((13.147)/(measasvolt))-0.42)      );
 // Use the equation to convert to a distance
      delay(50);
 // Pause 0.5s
      digitalWrite(calibcomp, HIGH);
// Set calibcomp signal high to PLC
      calibcompinternal = HIGH;
    // Set internal calibration complete signal high
                                  }
       if (calibcompinternal == HIGH){
     // Only allow for measurement to take place if calibrated
        for (int j = 0; j<=2; j++){
     // Initialise the For Loop for 3 sets of averages
         for (int i = 0; i <=9; i++) {
    // Initialise the For Loop for 10 measurements
          chk:
     // Pointer for a program jump
            float measurement = analogRead(sensor);
    // Read the value from the sensor
            float measasvolt = measurement * 0.0048828125;
     // Convert value from sensor to a voltage
```

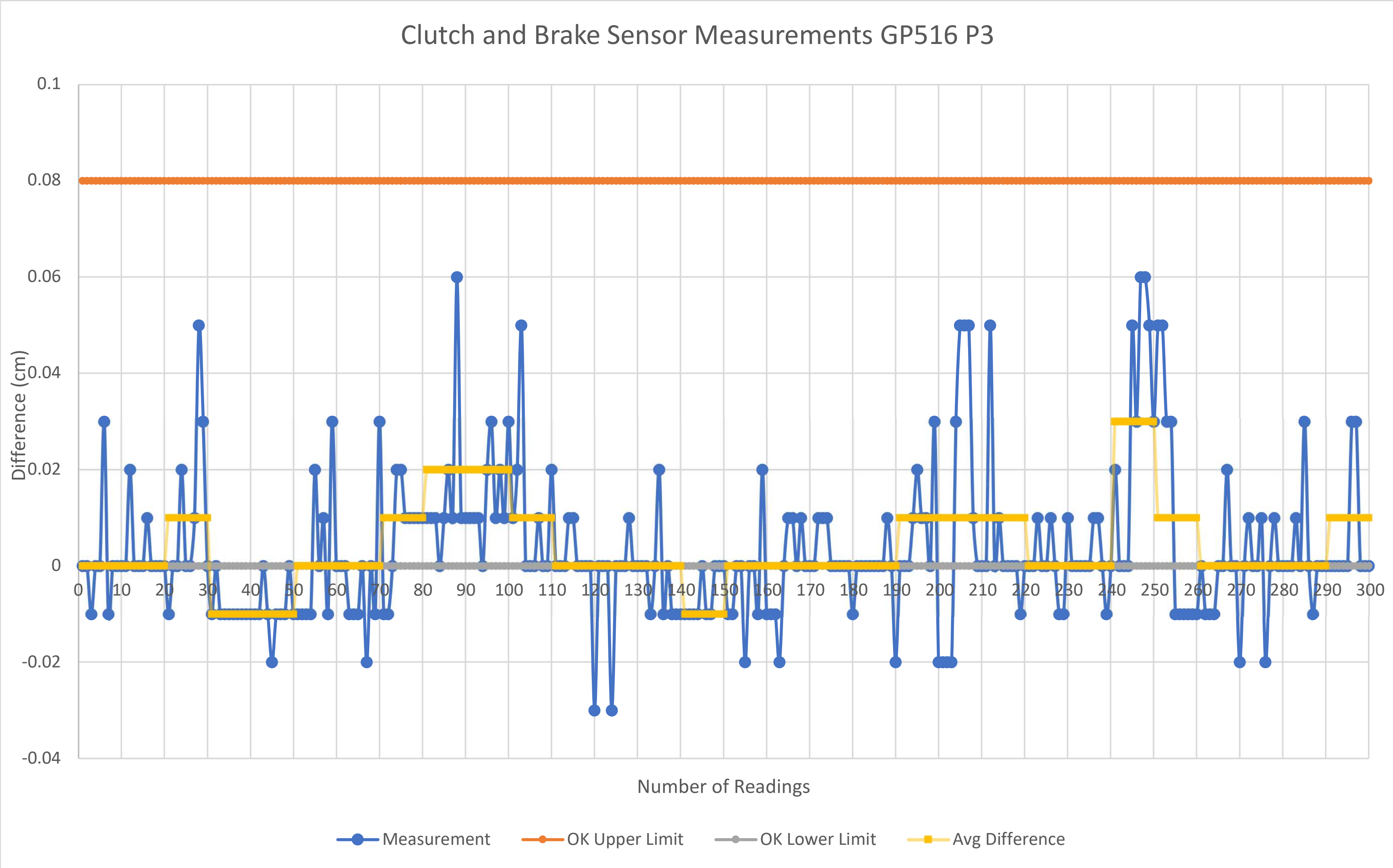
```
        float checkreading = ((calibdistance - (((13.147)/(measasvolt))-0.42)));
// Temporary storage for difference measurement
        delay(100);                                        // Pause 0.1s
          if (checkreading > 2){
// Check the reading to see if it is over 2
            goto chk;
// If this is the case then measure again
            delay(100);       }                            // Pause 0.1s

        if (i == 0)     {                                // The initial measurement

          difference0 = ((calibdistance - (((13.147)/(measasvolt))-0.42)));
// Find the difference by subtracting measured distance from calibration distance
            }

        if (i == 1)     {
// The second measurement

          difference1 = ((calibdistance - (((13.147)/(measasvolt))-0.42)));
// Find difference by subtracting measured distance from calibration distance
           }

        if (i == 2)     {
// The third measurement

          difference2 = ((calibdistance - (((13.147)/(measasvolt))-0.42)));
// Find difference by subtracting measured distance from calibration distance
            }

        if (i == 3)     {
// The fourth measurement

          difference3 = ((calibdistance - (((13.147)/(measasvolt))-0.42)));
// Find difference by subtracting measured distance from calibration distance
           }

        if (i == 4)     {
// The fifth measurement

          difference4 = ((calibdistance - (((13.147)/(measasvolt))-0.42)));
// Find difference by subtracting measured distance from calibration distance
          }
        if (i == 5)     {
// The sixth measurement

          difference5 = ((calibdistance - (((13.147)/(measasvolt))-0.42)));
// Find difference by subtracting measured distance from calibration distance
           }

        if (i == 6)     {
// The seventh measurement

          difference6 = ((calibdistance - (((13.147)/(measasvolt))-0.42)));
// Find difference by subtracting measured distance from calibration distance
           }

        if (i == 7)     {
// The eighth measurement

          difference7 = ((calibdistance - (((13.147)/(measasvolt))-0.42)));
```

```
// Find difference by subtracting measured distance from calibration distance
            }

        if (i == 8)      {
// The ninth measurement

          difference8 = ((calibdistance - (((13.147)/(measasvolt))-0.42)));
// Find difference by subtracting measured distance from calibration distance
            }

        if (i == 9)      {
// The tenth measurement

          difference9 = ((calibdistance - (((13.147)/(measasvolt))-0.42)));
// Find difference by subtracting measured distance from calibration distance
            }
                            }
      delay(50);
// Pause 0.05s

      avgdifference = ((difference0 + difference1 + difference2 + difference3 + difference4
  + difference5 + difference6 + difference7 + difference8 + difference9)/(10));
// Average the differences after 10 measurements
      avgdifferencefirstcomp = HIGH;
// The first average has been completed

      delay(50);
 // Pause 0.05s

        if (j == 0){
 // The first set of averages
        avgone = avgdifference; }
// The current average difference is copied to avgone hold

        delay(50);
// Pause 0.05s
        if (j == 1){
// The second set of averages
        avgtwo = avgdifference; }
// The current average difference is copied to avgtwo hold
        delay(50);
// Pause 0.05s

        if (j == 2){
// The third set of averages
        avgthree = avgdifference;}
// The current average difference is copied to avgthree hold
        delay(50);
// Pause 0.05s
                        }

     if(avgdifferencefirstcomp == HIGH){
// Check that a difference has been calculated before output

     if (avgone >= 0 && avgone <= 0.08 && avgtwo >= 0 && avgtwo <= 0.08 && avgthree
 >= 0 && avgthree <= 0.08){
// Check all average differences in a set range for OK

      digitalWrite(OK, HIGH);
// Measurement is OK - 1OK
```

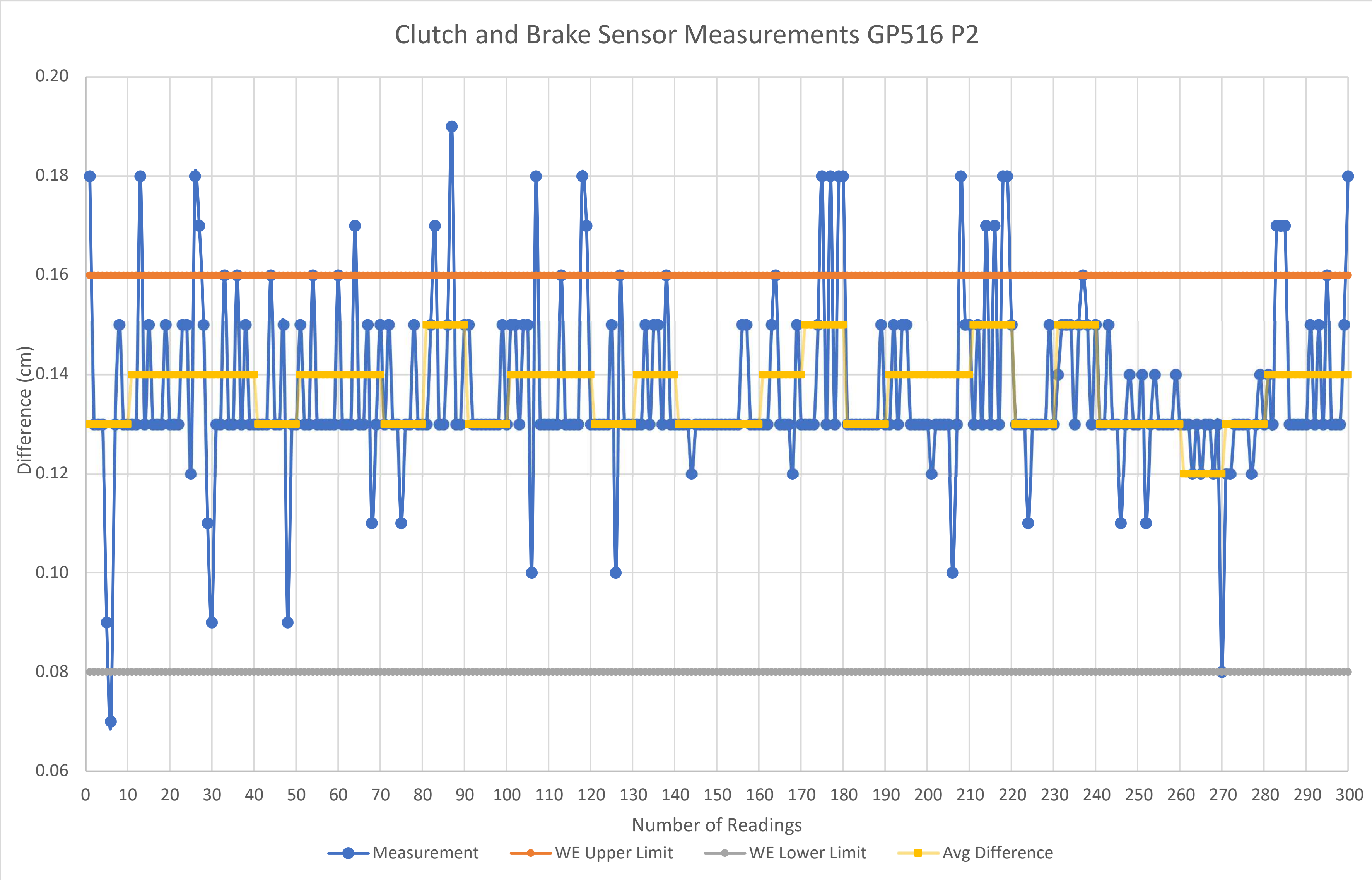
```
    digitalWrite(Wearing, LOW);
// Unused output set to off
    digitalWrite(NOK, LOW);
// Unused output set to off
    digitalWrite(LimitExceeded, LOW);
// The limits have not been exceeded
    const char text[] = "1OK";
// Set nRF transmission variable as '1OK'
    radio.write(&text, sizeof(text));
// Write the text[] variable over RF to the Central Graphical Device
    }

   else if (avgone > 0.08 && avgone <= 0.16 && avgtwo > 0.08 && avgtwo <= 0.16 &&
 avgthree > 0.08 && avgthree <= 0.16){
// Check all average differences in a set range for Wearing

    digitalWrite(OK, LOW);
// Unused output set to off
    digitalWrite(Wearing, HIGH);
// Measurement is Wearing - 1WE
    digitalWrite(NOK, LOW);
// Unused output set to off
    digitalWrite(LimitExceeded, LOW);
// The limits have not been exceeded
    const char text[] = "1WE";
 // Set nRF transmission variable as '1WE'
    radio.write(&text, sizeof(text));
// Write the text[] variable over RF to the Central Graphical Device
     }

   else if (avgone > 0.16 && avgone <= 1.00 && avgtwo > 0.16 && avgtwo <= 1.00 &&
 avgthree > 0.16 && avgthree <= 1.00){
// Check all average differences in a set range for NOK

    digitalWrite(OK, LOW);
// Unused output set to off
    digitalWrite(Wearing, LOW);
// Unused output set to off
    digitalWrite(NOK, HIGH);
// Measurement is NOK - 1NO
    digitalWrite(LimitExceeded, LOW);
// The limits have not been exceeded
    const char text[] = "1NO";
// Set nRF transmission variable as '1NO'
    radio.write(&text, sizeof(text));
// Write the text[] variable over RF to the Central Graphical Device
   }

   else if (avgone < 0 && avgtwo < 0 && avgthree < 0){
// Check if all average differences in a set are less than 0
    digitalWrite(OK, LOW);
// Unused output set to off
    digitalWrite(Wearing, LOW);
// Unused output set to off
    digitalWrite(NOK, LOW);
// Unused output set to off
    digitalWrite(LimitExceeded, LOW);
// The limits have been exceeded but it is classed as an error - 1ER
    const char text[] = "1ER";
// Set nRF transmission variable as '1ER'
```

```
        radio.write(&text, sizeof(text));
// Write the text[] variable over RF to the Central Graphical Device
        }
       else if (avgone > 1.00 && avgtwo > 1.00 && avgthree > 1.00) {
// Check if all average differences in a set are more than 1 - should never occur
        digitalWrite(OK, LOW);
// Unused output set to off
        digitalWrite(Wearing, LOW);
// Unused output set to off
        digitalWrite(NOK, LOW);
// Unused output set to off
        digitalWrite(LimitExceeded, HIGH);
// The physical limits have been exceeded - 1LE
        const char text[] = "1LE";
// Set nRF transmission variable as '1LE'
        radio.write(&text, sizeof(text));
// Write the text[] variable over RF to the Central Graphical Device
       }
       delay(100);
// Delay for transmission
       digitalWrite(lifebitout, LOW);
// Reset lifebit to PLC before next loop
       const char text[] = "";
// Empty variable ready for next iteration

      }
      }
      }
      }
      }
```

8.21 A21 C&B Implementation Test Results 1

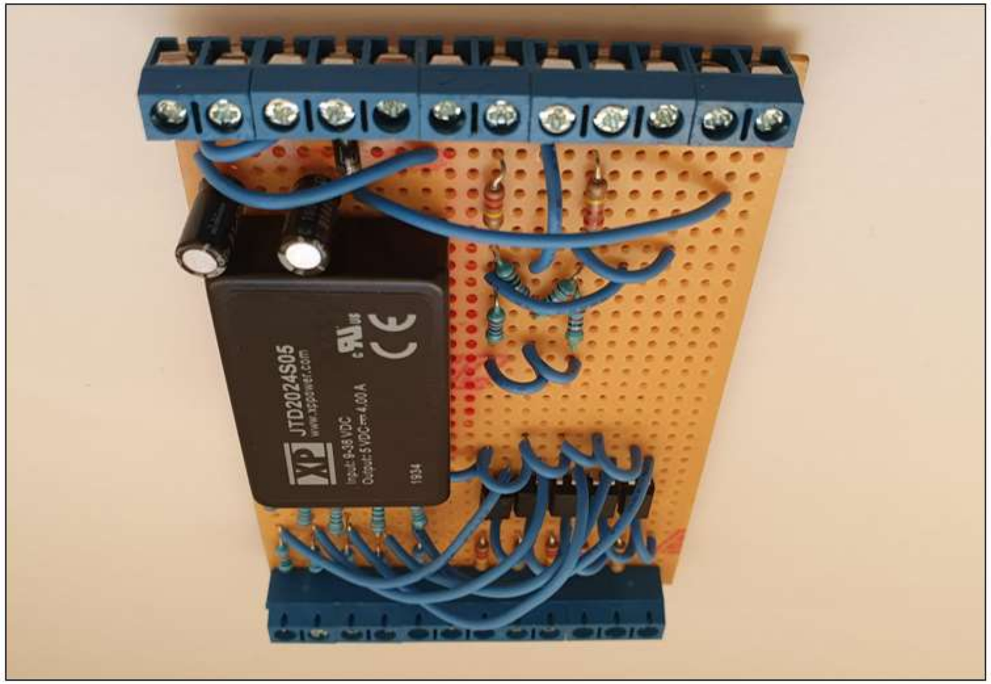


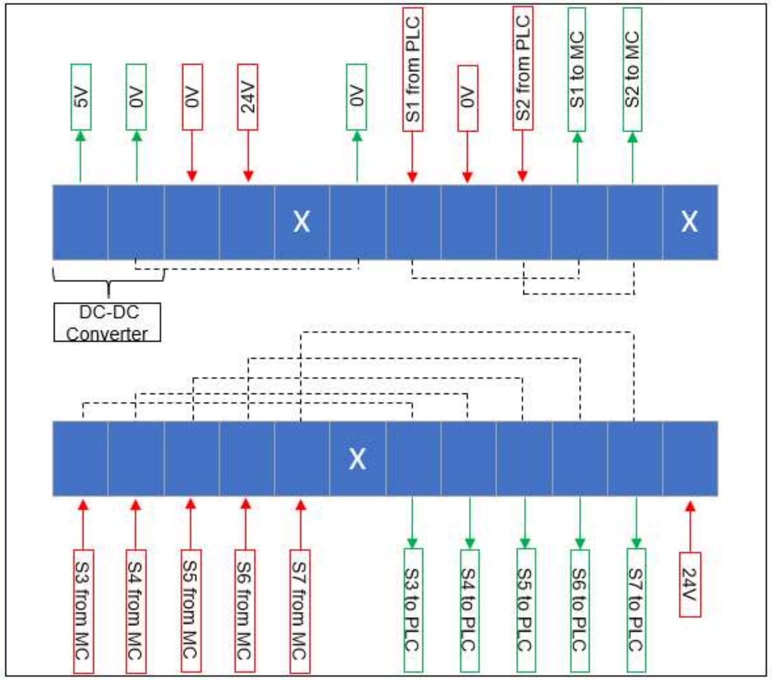
8.22 A22 C&B Implementation Test Results 2



8.23 A23 VA Interfacing Circuit Photograph and Terminal Designations

VA Interfacing Circuit Board

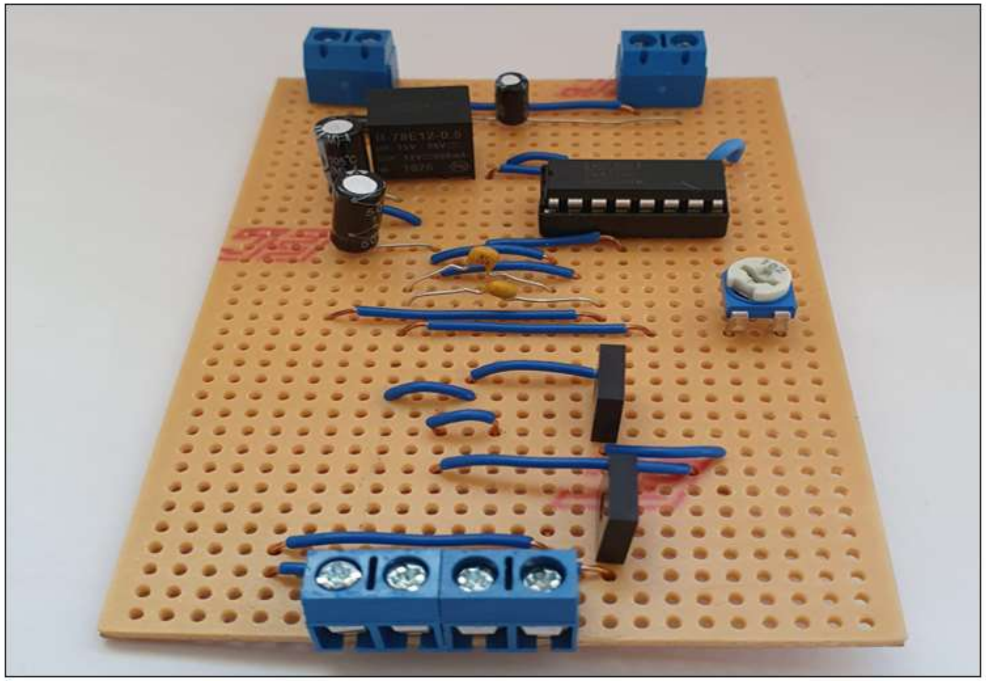


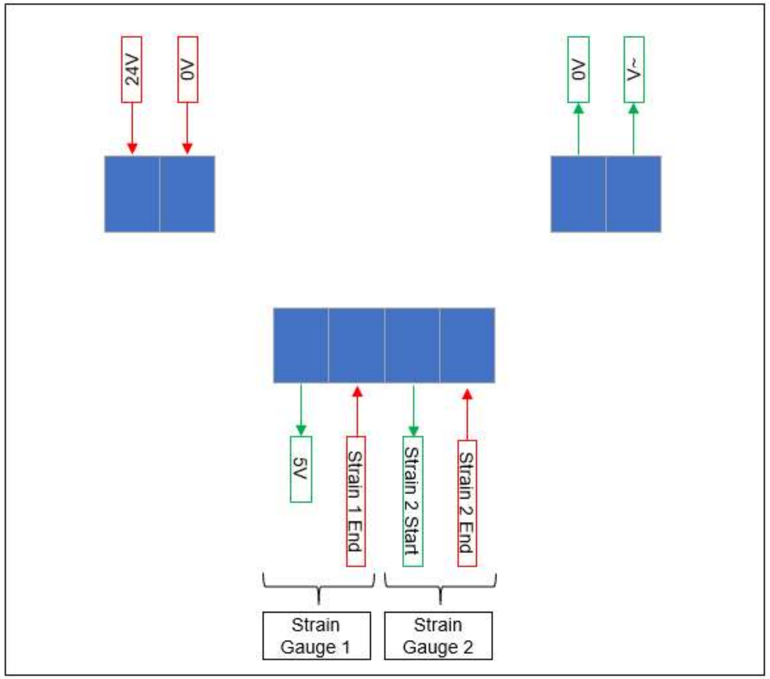


Information: blue blocks represent the terminals on the board, a red arrow and box is an input to the board whereas green is an output from the board. Sx is a signal. Dashed lines connect signals that are related.

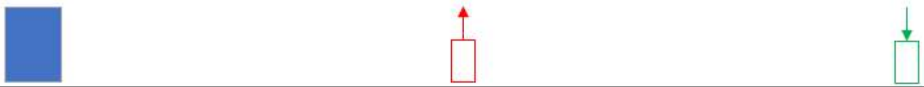
8.24 A24 NIPP Measuring Circuit Photograph and Terminal Designations

NIPP Control Circuit Board



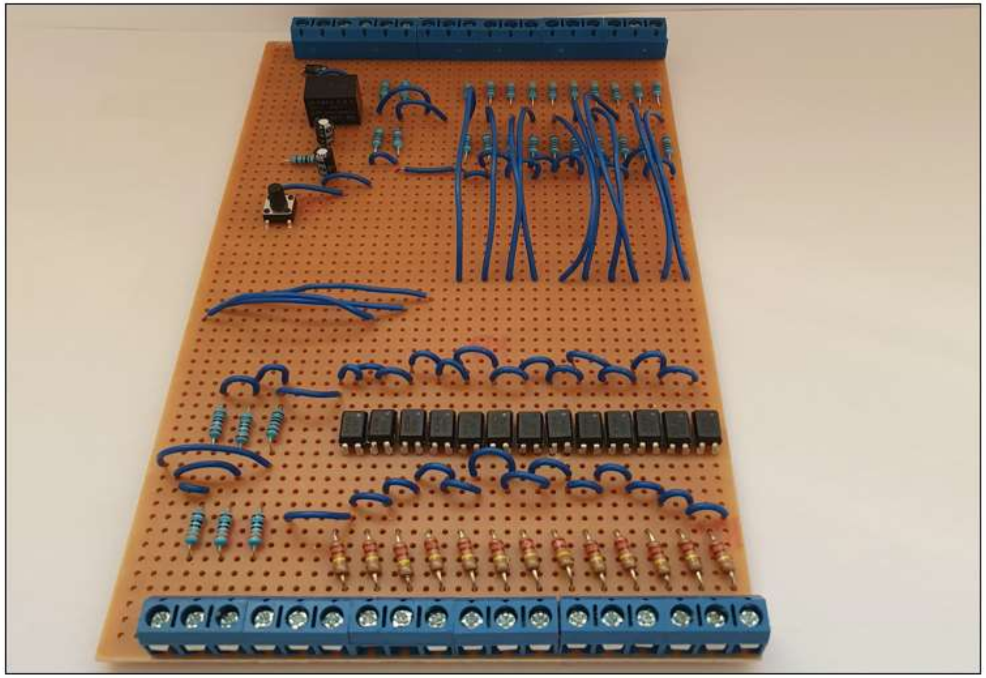


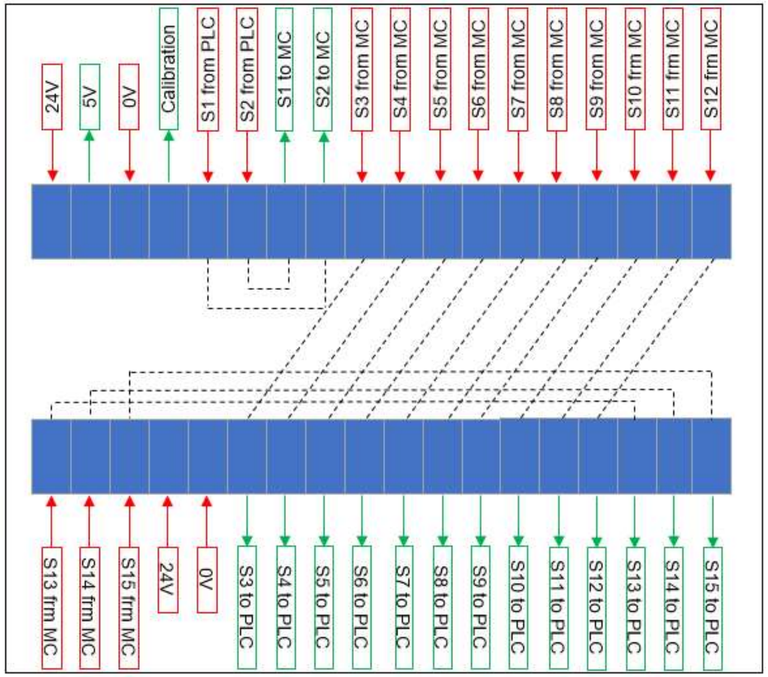
Information: blue blocks represent the terminals on the board, a red arrow and box is an input to the board whereas green is an output from the board. V~ is the varying output voltage.



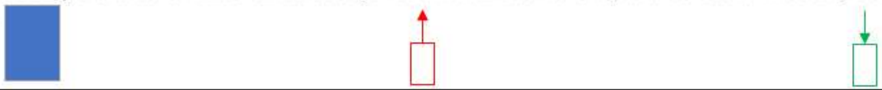
8.25 A25 NIPP Interfacing Circuit Photograph and Terminal Designations

NIPP Interfacing and Calibration Circuit Board





Information: blue blocks represent the terminals on the board, a red arrow and box is an input to the board whereas green is an output from the board. Sx is a signal. Dashed lines connect signals that are related.



8.26 A26 Complete NIPP Arduino Program

```
/* Non-Intrusive Pipe Pressure Monitoring */
/* nRF Libraries */
#include <SPI.h>                // Include the SPI library
#include <nRF24L01.h>                // nRF Library
#include <RF24.h>                // A further nRF24 library
/* nRF Setup */
RF24 radio(53, 48); // CE, CSN    // Declare pins for SPI communications
const byte address[6] = "00100";  // Setup nRF address for NIPP sensor
const char text[] = "";           // Transfer variable
/* Input and Output Declarations */
#define strain A0               // Analog input from the Strain Gauge sensor
const int calib = 30;           // Calibration button input to set a datum when pumps are off -
hold for
                                  5 seconds;
const int pumpson = 31;           // Pumps running signal from the PLC
const int lifebitin = 34;        // Life Bit signal from the PLC
const int lifebitout = 35;       // Life Bit signal to the PLC
const int calibcomp = 36;        // Calibration complete to allow measurement
const int bracketone = 32;       // Current condition is 0 to 10 bar signal to the PLC
const int brackettwo = 33;       // Current condition is 10 to 20 bar signal to the PLC
const int bracketthree = 37;     // Current condition is 20 to 30 bar signal to the PLC
const int bracketfour = 38;      // Current condition is 30 to 40 bar signal to the PLC
const int bracketfive = 39;      // Current condition is 40 to 50 bar signal to the PLC
const int bracketsix = 40;       // Current condition is 50 to 60 bar signal to the PLC
const int bracketseven = 41;     // Current condition is 60 to 70 bar signal to the PLC
const int bracketeight = 42;     // Current condition is 70 to 80 bar signal to the PLC
const int bracketnine = 43;      // Current condition is 80 to 90 bar signal to the PLC
const int bracketten = 44;       // Current condition is 90 to 100 bar signal to the PLC
const int LimitExceeded = 45;
 // The boundary limits have been exceeded signal to the PLC
/* Setup Local Variables */
float reading0 = 0.000;            // First hold area for the first reading
float reading1 = 0.000;            // Second hold area for the second reading
float reading2 = 0.000;            // Third hold area for the third reading
float reading3 = 0.000;            // Fourth hold area for the third reading
float reading4 = 0.000;            // Fifth hold area for the third reading
float reading5 = 0.000;            // Sixth hold area for the third reading
float reading6 = 0.000;            // Seventh hold area for the third reading
float reading7 = 0.000;            // Eighth hold area for the third reading
float reading8 = 0.000;            // Ninth hold area for the third reading
float reading9 = 0.000;            // Tenth hold area for the third reading
float checkreading;               // Check each reading before confirmation
float avgreading;                // Average of all the averages
float calibreading;              // The initial calibration reading at 0 bar pumps off
float avgone;                   // The first average from the first ten readings
float avgtwo;                   // The second average from the second ten readings
float avgthree;                  // The third average from the third ten readings
```

```

int calibcompinternal;              // Internal bit for calibration completion checking
int avgreadingfirstcomp;              // Checks the first avg reading has been taken


void setup() {

  pinMode(calib, INPUT);              // Set calib pin as an input
  pinMode(pumpson, INPUT);              // Set presstdc pin as an input
  pinMode(lifebitin, INPUT);          // Set lifebitin pin as an input
  pinMode(lifebitout, OUTPUT);         // Set lifebitout pin as an output
  pinMode(bracketone, OUTPUT);         // Set bracketone pin as an output
  pinMode(brackettwo, OUTPUT);         // Set brackettwo pin as an output
  pinMode(bracketthree, OUTPUT);        // Set bracketthree pin as an output
  pinMode(bracketfour, OUTPUT);        // Set bracketfour pin as an output
  pinMode(bracketfive, OUTPUT);        // Set bracketfive pin as an output
  pinMode(bracketsix, OUTPUT);         // Set bracketsix pin as an output
  pinMode(bracketseven, OUTPUT);        // Set bracketseven pin as an output
  pinMode(bracketeight, OUTPUT);       // Set bracketeight pin as an output
  pinMode(bracketnine, OUTPUT);        // Set bracketnine pin as an output
  pinMode(bracketten, OUTPUT);         // Set bracketten pin as an output
  pinMode(calibcomp, OUTPUT);          // Set calibcomp pin as an output
  pinMode(LimitExceeded, OUTPUT);       // Set LimitExceeded as an output
  digitalWrite(calibcomp, LOW);        // Set calibcomp low on power cycle

  radio.begin();                    // Initialise the nRF system
  radio.openWritingPipe(address);       // Open the writing pipe to address 00001
  radio.setPALevel(RF24_PA_MIN);        // Power Amplifier set to minimum level
  radio.stopListening();              // nRF is told to stop listening so it can write
          }

void loop() {

  if (digitalRead(lifebitin) == HIGH){
// Check lifebit from PLC is OK

    digitalWrite(lifebitout, HIGH);
// Send lifebit to PLC

    if (digitalRead(pumpson) == LOW)                 {
// Check that the pumps are OFF for 0 bar calibration

      if (digitalRead(calib) == HIGH){
// Check calibration button is pressed

              float measurement = analogRead(strain);
    // Read the value from the strain sensor circuit
              float measasvolt = measurement * 0.0048828125;
    // Convert value from sensor to a voltage
              calibreading =  (((measasvolt)-0.25)/(-0.0018));
    // Use the derived equation to convert to a pressure
              delay(50);
    // Pause 0.05s
              digitalWrite(calibcomp, HIGH);
     // Set calibcomp signal high to PLC
              calibcompinternal = HIGH;
    // Set internal calibration complete signal high
                                 }
                                              }

```

```
   if (digitalRead(pumpson) == HIGH)              {
// Only allow for readings if pumps are on
    if (calibcompinternal == HIGH){
// Only allow for reading to take place if calibrated
     for (int j = 0; j<=2; j++){
      for (int i = 0; i <=9; i++) {
// Initialise For Loop for 3 readings
       chk:
        float measurement = analogRead(strain);
// Read the value from the sensor
        float measasvolt = measurement * 0.0048828125;
 // Convert value from sensor to a voltage
        float checkreading = (((measasvolt)-0.25)/(-0.0018))- calibreading;
        delay(100);
          if (checkreading > 150){
            goto chk;
            delay(100);        }
        if (i == 0)     {
// The first reading
          reading0 = (((measasvolt)-0.25)/(-0.0018)) - calibreading;
// Find the pressure and offset the 0 bar calibration
             }
        if (i == 1)     {
// The second reading
          reading1 = (((measasvolt)-0.25)/(-0.0018)) - calibreading;
// Find the pressure and offset the 0 bar calibration
             }
        if (i == 2)     {
 // The third reading
          reading2 = (((measasvolt)-0.25)/(-0.0018)) - calibreading;
// Find the pressure and offset the 0 bar calibration
             }
        if (i == 3)     {
// The fourth reading
          reading3 = (((measasvolt)-0.25)/(-0.0018)) - calibreading;
 // Find the pressure and offset the 0 bar calibration
             }
        if (i == 4)     {
// The fifth reading
          reading4 = (((measasvolt)-0.25)/(-0.0018)) - calibreading;
// Find the pressure and offset the 0 bar calibration
             }
        if (i == 5)     {
// The sixth reading
          reading5 = (((measasvolt)-0.25)/(-0.0018)) - calibreading;
        // Find the pressure and offset the 0 bar calibration
             }
        if (i == 6)     {
// The seventh reading
          reading6 = (((measasvolt)-0.25)/(-0.0018)) - calibreading;
// Find the pressure and offset the 0 bar calibration
```

```
            }

        if (i == 7)     {
// The eighth reading
            reading7 = (((measasvolt)-0.25)/(-0.0018)) - calibreading;

// Find the pressure and offset the 0 bar calibration
            }

        if (i == 8)     {
// The ninth reading
            reading8 = (((measasvolt)-0.25)/(-0.0018)) - calibreading;

// Find the pressure and offset the 0 bar calibration
            }

        if (i == 9)     {
// The tenth reading
            reading9 = (((measasvolt)-0.25)/(-0.0018)) - calibreading;

// Find the pressure and offset the 0 bar calibration
            }
                        }
    delay(50);
// Pause 0.05s

    avgreading = ((reading0 + reading1 + reading2 + reading3 + reading4 + reading5 +
reading6 + reading7 + reading8 + reading9)/10);
// Average the readings after 10 measurements
    avgreadingfirstcomp = HIGH;

    delay(50);
// Pause 0.05s

        if (j == 0){
// The first set of averages
        avgone = avgreading; }
// The current average difference is copied to avgone hold
        delay(50);
// Pause 0.05s
        if (j == 1){
// The second set of averages
        avgtwo = avgreading; }
// The current average difference is copied to avgtwo hold
        delay(50);
// Pause 0.05s
        if (j == 2){
// The third set of averages
        avgthree = avgreading;}
// The current average difference is copied to avgthree hold
        delay(50);
// Pause 0.05s
                        }

    if(avgreadingfirstcomp == HIGH){
// Check that a read has been calculated before output

    if (avgone >= 0 && avgone < 10 && avgtwo >= 0 && avgtwo < 10 && avgthree >= 0
 && avgthree < 10){
// Check all average differences in a set range for 0 to 10 bar
```

```
        digitalWrite(bracketone,    HIGH);
// Measurement is in Bracket 1
        digitalWrite(brackettwo,    LOW);
// Measurement is not in Bracket 2
        digitalWrite(bracketthree,  LOW);
// Measurement is not in Bracket 3
        digitalWrite(bracketfour,   LOW);
// Measurement is not in Bracket 4
        digitalWrite(bracketfive,   LOW);
// Measurement is not in Bracket 5
        digitalWrite(bracketsix,    LOW);
// Measurement is not in Bracket 6
        digitalWrite(bracketseven,  LOW);
// Measurement is not in Bracket 7
        digitalWrite(bracketeight,  LOW);
// Measurement is not in Bracket 8
        digitalWrite(bracketnine,   LOW);
// Measurement is not in Bracket 9
        digitalWrite(bracketten,    LOW);
// Measurement is not in Bracket 10
        digitalWrite(LimitExceeded, LOW);
// The limits have not been exceeded
        const char text[] = "301";
// Set nRF transmission variable as '301'
        radio.write(&text, sizeof(text));
// Write the text[] variable over RF to the Central Graphical Device
                                    }
       else if (avgone >= 10 && avgone < 20 && avgtwo >= 10 && avgtwo < 20 &&
avgthree >= 10 && avgthree < 20){
// Check all average differences in a set range for 10 to 20 bar

        digitalWrite(bracketone,    LOW);
// Measurement is not in Bracket 1
        digitalWrite(brackettwo,    HIGH);
// Measurement is in Bracket 2
        digitalWrite(bracketthree,  LOW);
// Measurement is not in Bracket 3
        digitalWrite(bracketfour,   LOW);
// Measurement is not in Bracket 4
        digitalWrite(bracketfive,   LOW);
// Measurement is not in Bracket 5
        digitalWrite(bracketsix,    LOW);
// Measurement is not in Bracket 6
        digitalWrite(bracketseven,  LOW);
// Measurement is not in Bracket 7
        digitalWrite(bracketeight,  LOW);
// Measurement is not in Bracket 8
        digitalWrite(bracketnine,   LOW);
// Measurement is not in Bracket 9
        digitalWrite(bracketten,    LOW);
// Measurement is not in Bracket 10
        digitalWrite(LimitExceeded, LOW);
// The limits have not been exceeded
        const char text[] = "302";
// Set nRF transmission variable as '302'
        radio.write(&text, sizeof(text));
// Write the text[] variable over RF to the Central Graphical Device
                                    }
       else if (avgone >= 20 && avgone < 30 && avgtwo >= 20 && avgtwo < 30 &&
```

```
avgthree >= 20 && avgthree < 30){
// Check all average differences in a set range for 20 to 30 bar

    digitalWrite(bracketone,    LOW);
// Measurement is not in Bracket 1
    digitalWrite(brackettwo,    LOW);
// Measurement is not in Bracket 2
    digitalWrite(bracketthree,  HIGH);
// Measurement is in Bracket 3
    digitalWrite(bracketfour,   LOW);
// Measurement is not in Bracket 4
    digitalWrite(bracketfive,   LOW);
// Measurement is not in Bracket 5
    digitalWrite(bracketsix,    LOW);
// Measurement is not in Bracket 6
    digitalWrite(bracketseven,  LOW);
// Measurement is not in Bracket 7
    digitalWrite(bracketeight,  LOW);
// Measurement is not in Bracket 8
    digitalWrite(bracketnine,   LOW);
// Measurement is not in Bracket 9
    digitalWrite(bracketten,    LOW);
// Measurement is not in Bracket 10
    digitalWrite(LimitExceeded, LOW);
// The limits have not been exceeded
    const char text[] = "303";
// Set nRF transmission variable as '303'
    radio.write(&text, sizeof(text));
// Write the text[] variable over RF to the Central Graphical Device
                                          }
   else if (avgone >= 30 && avgone < 40 && avgtwo >= 30 && avgtwo < 40 &&
avgthree >= 30 && avgthree < 40){
// Check all average differences in a set range for 30 to 40 bar

    digitalWrite(bracketone,    LOW);
// Measurement is not in Bracket 1
    digitalWrite(brackettwo,    LOW);
// Measurement is not in Bracket 2
    digitalWrite(bracketthree,  LOW);
// Measurement is not in Bracket 3
    digitalWrite(bracketfour,   HIGH);
// Measurement is in Bracket 4
    digitalWrite(bracketfive,   LOW);
// Measurement is not in Bracket 5
    digitalWrite(bracketsix,    LOW);
// Measurement is not in Bracket 6
    digitalWrite(bracketseven,  LOW);
// Measurement is not in Bracket 7
    digitalWrite(bracketeight,  LOW);
// Measurement is not in Bracket 8
    digitalWrite(bracketnine,   LOW);
// Measurement is not in Bracket 9
    digitalWrite(bracketten,    LOW);
// Measurement is not in Bracket 10
    digitalWrite(LimitExceeded, LOW);
// The limits have not been exceeded
    const char text[] = "304";
// Set nRF transmission variable as '304'
    radio.write(&text, sizeof(text));
// Write the text[] variable over RF to the Central Graphical Device
```

```
                                            }

        else if (avgone >= 40 && avgone < 50 && avgtwo >= 40 && avgtwo <50 &&
avgthree >= 40 && avgthree < 50){
// Check all average differences in a set range for 40 to 50 bar

        digitalWrite(bracketone,    LOW);
// Measurement is not in Bracket 1
        digitalWrite(brackettwo,    LOW);
// Measurement is not in Bracket 2
        digitalWrite(bracketthree,  LOW);
 // Measurement is not in Bracket 3
        digitalWrite(bracketfour,   LOW);
// Measurement is not in Bracket 4
        digitalWrite(bracketfive,   HIGH);
// Measurement is in Bracket 5
        digitalWrite(bracketsix,    LOW);
// Measurement is not in Bracket 6
        digitalWrite(bracketseven,  LOW);
// Measurement is not in Bracket 7
        digitalWrite(bracketeight,  LOW);
// Measurement is not in Bracket 8
        digitalWrite(bracketnine,   LOW);
// Measurement is not in Bracket 9
        digitalWrite(bracketten,    LOW);
// Measurement is not in Bracket 10
        digitalWrite(LimitExceeded, LOW);
// The limits have not been exceeded
        const char text[] = "305";
// Set nRF transmission variable as '305'
        radio.write(&text, sizeof(text));
// Write the text[] variable over RF to the Central Graphical Device
                                                                            }

        else if (avgone >= 50 && avgone < 60 && avgtwo >= 50 && avgtwo < 60 &&
avgthree >= 50 && avgthree < 60){
// Check all average differences in a set range for 50 to 60 bar

        digitalWrite(bracketone,    LOW);
// Measurement is not in Bracket 1
        digitalWrite(brackettwo,    LOW);
// Measurement is not in Bracket 2
        digitalWrite(bracketthree,  LOW);
// Measurement is not in Bracket 3
        digitalWrite(bracketfour,   LOW);
// Measurement is not in Bracket 4
        digitalWrite(bracketfive,   LOW);
// Measurement is not in Bracket 5
        digitalWrite(bracketsix,    HIGH);
// Measurement is in Bracket 6
        digitalWrite(bracketseven,  LOW);
// Measurement is not in Bracket 7
        digitalWrite(bracketeight,  LOW);
// Measurement is not in Bracket 8
        digitalWrite(bracketnine,   LOW);
// Measurement is not in Bracket 9
        digitalWrite(bracketten,    LOW);
// Measurement is not in Bracket 10
        digitalWrite(LimitExceeded, LOW);
// The limits have not been exceeded
```

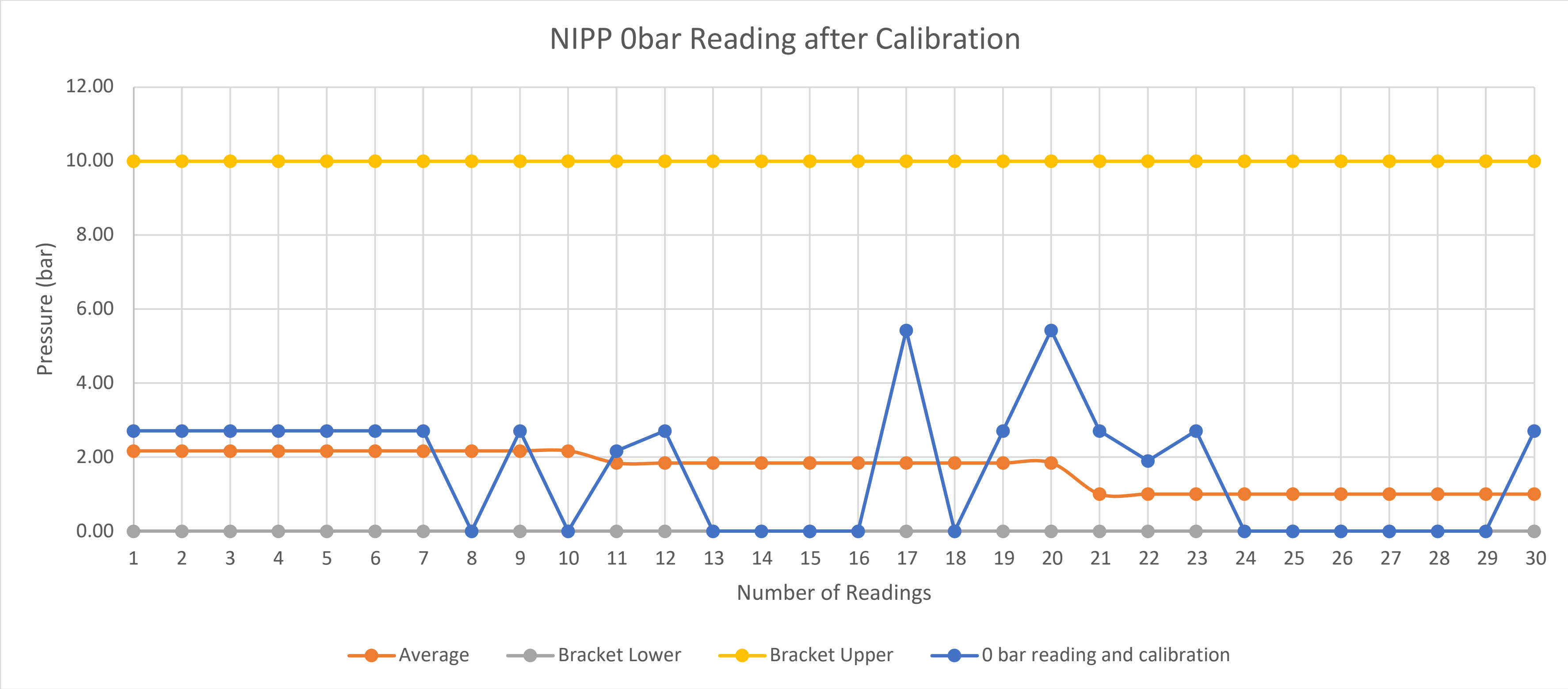
```
    const char text[] = "306";
// Set nRF transmission variable as '306'
    radio.write(&text, sizeof(text));
// Write the text[] variable over RF to the Central Graphical Device
                                                          }

   else if (avgone >= 60 && avgone < 70 && avgtwo >= 60 && avgtwo < 70 &&
avgthree >= 60 && avgthree < 70){
// Check all average differences in a set range for 60 to 70 bar

    digitalWrite(bracketone,    LOW);
// Measurement is not in Bracket 1
    digitalWrite(brackettwo,    LOW);
// Measurement is not in Bracket 2
    digitalWrite(bracketthree,  LOW);
// Measurement is not in Bracket 3
    digitalWrite(bracketfour,   LOW);
// Measurement is not in Bracket 4
    digitalWrite(bracketfive,   LOW);
// Measurement is not in Bracket 5
    digitalWrite(bracketsix,    LOW);
// Measurement is not in Bracket 6
    digitalWrite(bracketseven,  HIGH);
// Measurement is in Bracket 7
    digitalWrite(bracketeight,  LOW);
// Measurement is not in Bracket 8
    digitalWrite(bracketnine,   LOW);
// Measurement is not in Bracket 9
    digitalWrite(bracketten,    LOW);
// Measurement is not in Bracket 10
    digitalWrite(LimitExceeded, LOW);
// The limits have not been exceeded
    const char text[] = "307";
// Set nRF transmission variable as '307'
    radio.write(&text, sizeof(text));
// Write the text[] variable over RF to the Central Graphical Device
     }

   else if (avgone >= 70 && avgone < 80 && avgtwo >= 70 && avgtwo < 80 &&
avgthree >= 70 && avgthree < 80){
// Check all average differences in a set range for 70 to 80 bar

    digitalWrite(bracketone,    LOW);
// Measurement is not in Bracket 1
    digitalWrite(brackettwo,    LOW);
// Measurement is not in Bracket 2
    digitalWrite(bracketthree,  LOW);
// Measurement is not in Bracket 3
    digitalWrite(bracketfour,   LOW);
// Measurement is not in Bracket 4
    digitalWrite(bracketfive,   LOW);
// Measurement is not in Bracket 5
    digitalWrite(bracketsix,    LOW);
// Measurement is not in Bracket 6
    digitalWrite(bracketseven,  LOW);
// Measurement is not in Bracket 7
    digitalWrite(bracketeight,  HIGH);
// Measurement is in Bracket 8
    digitalWrite(bracketnine,   LOW);
// Measurement is not in Bracket 9
```

```
    digitalWrite(bracketten,    LOW);
// Measurement is not in Bracket 10
    digitalWrite(LimitExceeded, LOW);
// The limits have not been exceeded
    const char text[] = "308";
// Set nRF transmission variable as '308'
    radio.write(&text, sizeof(text));
// Write the text[] variable over RF to the Central Graphical Device
    }

   else if (avgone >= 80 && avgone < 90 && avgtwo >= 80 && avgtwo < 90 &&
avgthree >= 80 && avgthree < 90){
// Check all average differences in a set range for 80 to 90 bar

    digitalWrite(bracketone,    LOW);
// Measurement is not in Bracket 1
    digitalWrite(brackettwo,    LOW);
// Measurement is not in Bracket 2
    digitalWrite(bracketthree,  LOW);
// Measurement is not in Bracket 3
    digitalWrite(bracketfour,   LOW);
 // Measurement is not in Bracket 4
    digitalWrite(bracketfive,   LOW);
// Measurement is not in Bracket 5
    digitalWrite(bracketsix,    LOW);
// Measurement is not in Bracket 6
    digitalWrite(bracketseven,  LOW);
// Measurement is not in Bracket 7
    digitalWrite(bracketeight,  LOW);
// Measurement is not in Bracket 8
    digitalWrite(bracketnine,   HIGH);
// Measurement is in Bracket 9
    digitalWrite(bracketten,    LOW);
// Measurement is not in Bracket 10
    digitalWrite(LimitExceeded, LOW);
// The limits have not been exceeded
    const char text[] = "309";
// Set nRF transmission variable as '309'
    radio.write(&text, sizeof(text));
// Write the text[] variable over RF to the Central Graphical Device
 }

   else if (avgone >= 90 && avgone <= 100 && avgtwo >= 90 && avgtwo <= 100 &&
avgthree >= 90 && avgthree <= 100){
// Check all average differences in a set range for 90 to 100 bar

    digitalWrite(bracketone,    LOW);
// Measurement is not in Bracket 1
    digitalWrite(brackettwo,    LOW);
// Measurement is not in Bracket 2
    digitalWrite(bracketthree,  LOW);
// Measurement is not in Bracket 3
    digitalWrite(bracketfour,   LOW);
// Measurement is not in Bracket 4
    digitalWrite(bracketfive,   LOW);
// Measurement is not in Bracket 5
    digitalWrite(bracketsix,    LOW);
// Measurement is not in Bracket 6
    digitalWrite(bracketseven,  LOW);
// Measurement is not in Bracket 7
```

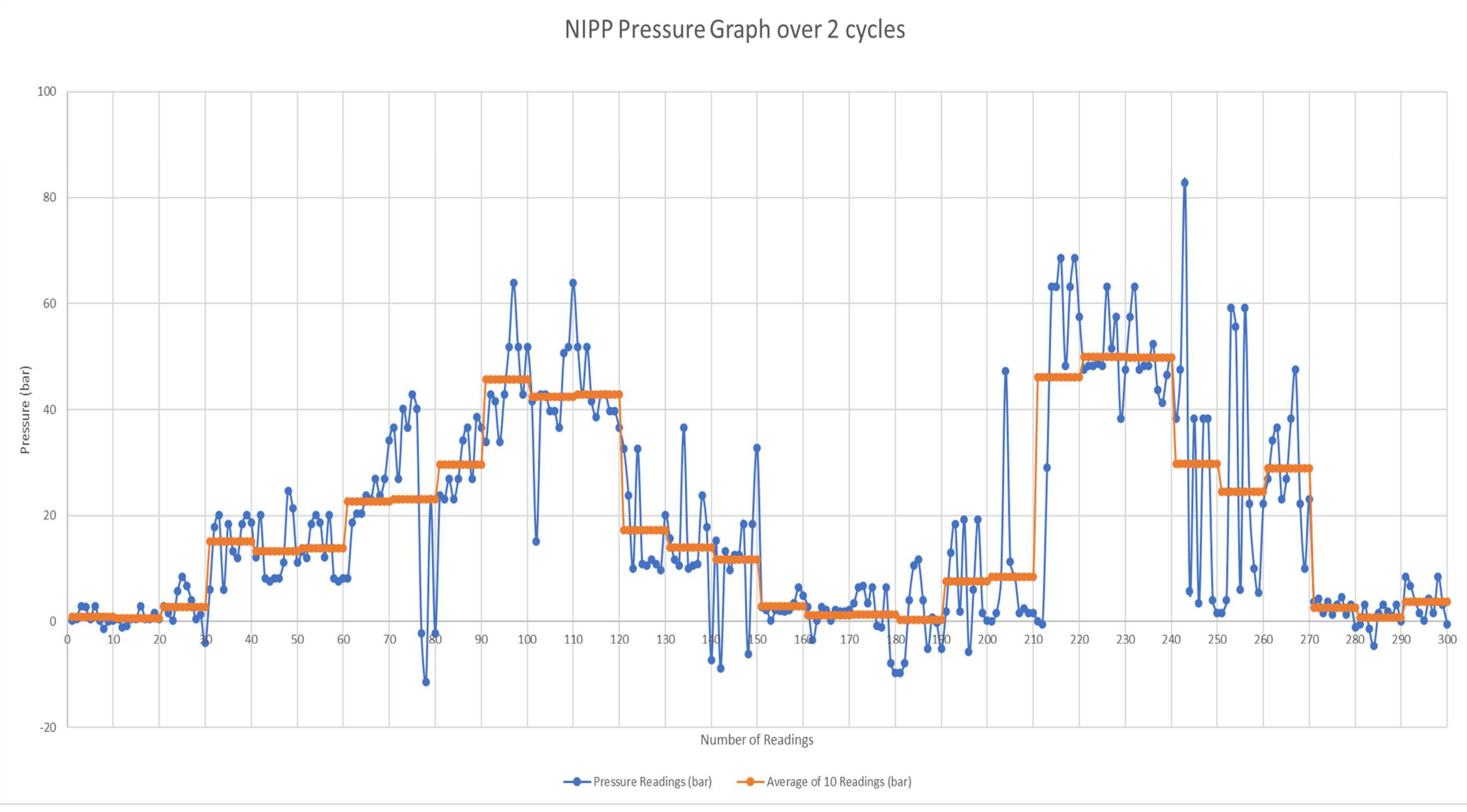
```
      digitalWrite(bracketeight,  LOW);
// Measurement is not in Bracket 8
      digitalWrite(bracketnine,   LOW);
// Measurement is not in Bracket 9
      digitalWrite(bracketten,    HIGH);
// Measurement is in Bracket 10
      digitalWrite(LimitExceeded, LOW);
// The limits have not been exceeded
      const char text[] = "310";
// Set nRF transmission variable as '310'
      radio.write(&text, sizeof(text));
// Write the text[] variable over RF to the Central Graphical Device
        }
     else if (avgone < 0 && avgtwo < 0 && avgthree < 0){
      digitalWrite(bracketone,    LOW);
// Measurement is not in Bracket 1
      digitalWrite(brackettwo,    LOW);
// Measurement is not in Bracket 2
      digitalWrite(bracketthree,  LOW);
// Measurement is not in Bracket 3
      digitalWrite(bracketfour,   LOW);
// Measurement is not in Bracket 4
      digitalWrite(bracketfive,   LOW);
// Measurement is not in Bracket 5
      digitalWrite(bracketsix,    LOW);
// Measurement is not in Bracket 6
      digitalWrite(bracketseven,  LOW);
// Measurement is not in Bracket 7
      digitalWrite(bracketeight,  LOW);
// Measurement is not in Bracket 8
      digitalWrite(bracketnine,   LOW);
// Measurement is not in Bracket 9
      digitalWrite(bracketten,    LOW);
// Measurement is not in Bracket 10
      digitalWrite(LimitExceeded, LOW);
// The limits have been exceeded but it is classified as an error
      const char text[] = "3ER";
// Set nRF transmission variable as '3ER'
      radio.write(&text, sizeof(text));
// Write the text[] variable over RF to the Central Graphical Device
        }
     else if (avgone > 100 && avgtwo > 100 && avgthree > 100){
      digitalWrite(bracketone,    LOW);
// Measurement is not in Bracket 1
      digitalWrite(brackettwo,    LOW);
// Measurement is not in Bracket 2
      digitalWrite(bracketthree,  LOW);
// Measurement is not in Bracket 3
      digitalWrite(bracketfour,   LOW);
// Measurement is not in Bracket 4
      digitalWrite(bracketfive,   LOW);
// Measurement is not in Bracket 5
      digitalWrite(bracketsix,    LOW);
// Measurement is not in Bracket 6
      digitalWrite(bracketseven,  LOW);
// Measurement is not in Bracket 7
      digitalWrite(bracketeight,  LOW);
// Measurement is not in Bracket 8
      digitalWrite(bracketnine,   LOW);
// Measurement is not in Bracket 9
```

```
        digitalWrite(bracketten,    LOW);
    // Measurement is not in Bracket 10
        digitalWrite(LimitExceeded, HIGH);
     // The limits have been exceeded
        const char text[] = "3LE";
    // Set nRF transmission variable as '3LE'
        radio.write(&text, sizeof(text));
    // Write the text[] variable over RF to the Central Graphical Device
       }
       delay(100);
    // Delay for transmission
       digitalWrite(lifebitout, LOW);
    // Reset lifebit to PLC before next loop
       const char text[] = "";
    // Empty variable ready for next iteration
       }
       }
       }
       }    }
```

8.27 A27 NIPP Pressure Readings Graph 1

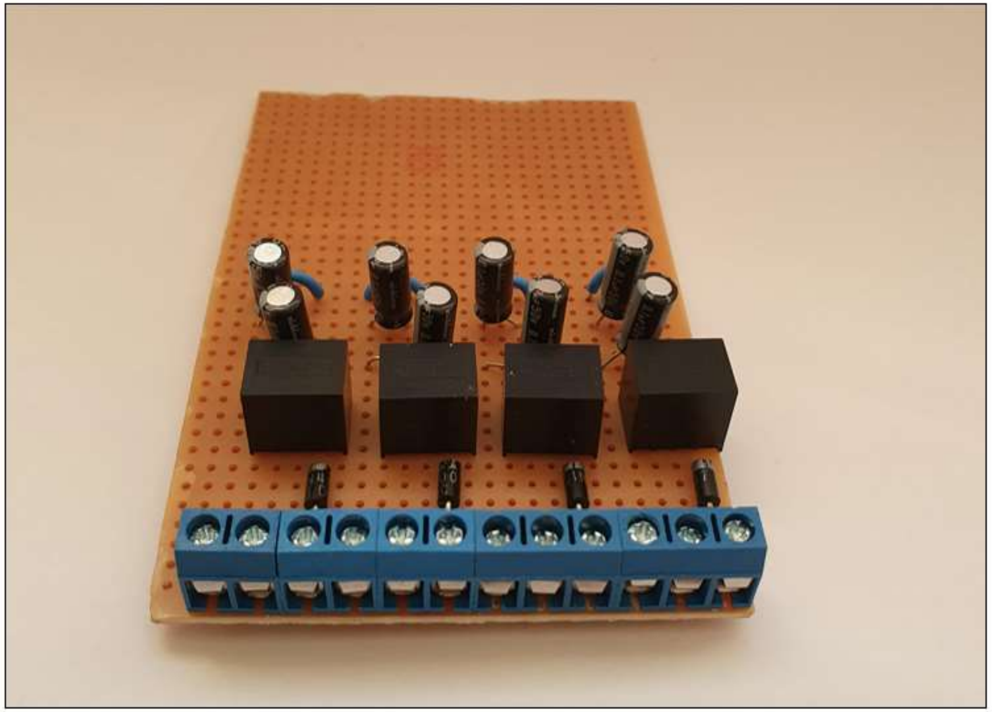


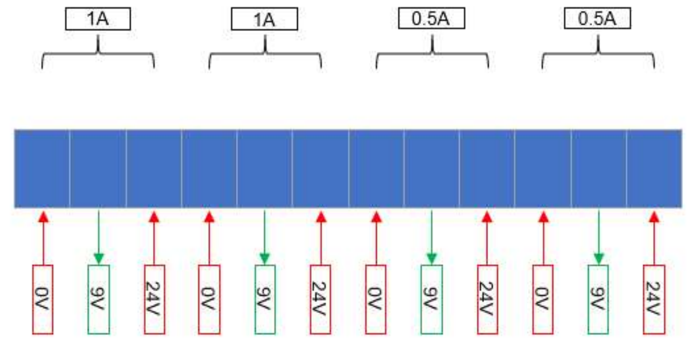
8.28 A28 NIPP Pressure Readings Graph 2



8.29 A29 9V Regulator Circuit Photograph and Terminal Designations

9V Voltage Regulator Circuit Board





Information: blue blocks represent the terminals on the board, a red arrow and box is an input to the board whereas green is an output from the board.