A Real Time Fault Detection and Diagnosis System for Automotive Applications

A Real Time Fault Detection and Diagnosis System for Automotive Applications

By

Ahmed Doghri

A THESIS

SUBMITTED TO THE DEPARTMENT OF MECHANICAL ENGINEERING

AND THE SCHOOL OF GRADUATE STUDIES

OF MCMASTER UNIVERSITY

IN PARTIAL FULFILMENT OF THE REQUIREMENTS

FOR THE DEGREE

MASTER OF APPLIED SCIENCE

MASTER OF APPLIED SCIENCE (2019) MCMASTER UNIVERISTY

MECHANICAL ENGINEERING HAMILTON, ONTARIO, CANADA

TITLE: A Real Time Fault Detection and Diagnosis System for Automotive Applications

AUTHOR: Ahmed Doghri

B.Sc. in Electrical Engineering

KAIST University, Daejeon, South Korea

SUPERVISOR: Professor Saeid Habibi

NUMBER OF PAGES: xxi, 223

ABSTRACT

Since its inception in the nineteenth century, the Internal Combustion Engine (ICE) remains the most prevalent technology in transportation systems to date. In order to minimize emissions, it is important that ICE is operated according to its optimized design conditions. As such, condition monitoring and Fault Detection and Diagnosis (FDD) tools can play an important role in detecting conditions that would affect the operability of the engine. In this research, different signal-based Fault Detection and Diagnosis (FDD) techniques are researched and implemented for fault condition monitoring of ICE. The implementation of prognostics for the engine in an automated form has important consequences that include cost savings, increased reliability, reduction of GHG emissions, better safety, and extended life for the vehicle.

In this research, in order to carry out FDD onboard, a low-cost and flexible internet-based data-acquisition system (DAQ) was designed and implemented. The main part of the system is an embedded hardware running a full desktop version of Linux. This sensory system leverages the positive aspects of both real-time and general-purpose architectures to ensure engine monitoring at high sampling rates. Unlike other commercial DAQ systems, the software of this device is open-source, free of charge, and highly expandable to suit other FDD applications.

In addition to data collection at high sampling rates, the FDD system includes advanced FDD strategies. The Fault Detection and Diagnosis strategies considered use a combination of Fourier Transforms (FT), Wavelet Transforms (WT), and Principal Component Analysis (PCA). Meanwhile, Fault Classification was carried using Neural Networks consisting of the Multi-Layer Perceptron (MLP). Three strategies were comparatively considered for the training of the Neural Network (NN), namely the Levenberg-Marquardt (LM), the Extended Kalman Filter (EKF), and

the Smooth Variable Structure Filter (SVSF) techniques. The proposed FDD system was able to achieve 100% accuracy in classifying a set of engine faults.

ACKNOWLEDGEMENTS

I would like to start by dedicating special thanks to my supervisor Dr. Habibi for his valuable guidance and advice throughout my research in the past two years and in the completion of this thesis.

I am also grateful to the entire CMHT team for taking up the role of my family here in Canada. In particular, I would like to acknowledge Mr. Cam Fisher's constructive support and assistance in designing the data acquisition system for this research, Dr. Mahmoud Ismail for explaining the different mathematical tricks involved in his signal processing algorithm (IEMSPCA), and Mr. Feng Yifei for explaining the gist of neural network-based fault classification algorithms.

Finally, I am would like to extend my warm regards to thank my family to whom I owe all of my accomplishments.

List of Abbreviations

ADC	Analog to Digital Converter
AEM	Abnormal Event Management
AQA	Air Quality Agreement
ANN	Artificial Neural Network
A/D	Analog to Digital
BDC	Bottom-Dead Center
CAD	Crank Angle Domain
CAFE	Corporate Average Fuel Economy
CAN	Controller Area Network
CASA	Crank Angle Speed Analysis
CMHT	Centre for Mechatronics and Hybrid Technologies
CNI	Combustion Noise Intensity
CPU	Central Processing Unit
CSTR	Continuous Stirred Tank Reactor
CTFT	Continuous Time Fourier Transform
CTC	Connection Technology Center Inc.
CVA	Canonical Variate Analysis
CWT	Continuous Wavelet Transform
DAQ	Data Acquisition System
DC	Direct Current
DCG	Dynamic Clock Gating
DFT	Discrete Fourier Transform
DFS	Dynamic Frequency Scaling
DPCA	Dynamic Principle Component Analysis
DTFT	Discrete Time Fourier Transform
DWT	Discrete Wavelet Transform
ECT	Electronic Control Thermal
ECU	Engine Control Unit
EKF	Extended Kalman Filter
ENOB	Effective Number Of Bits
EPA	Environment Protection Agency
FDD	Fault Detection and Diagnosis
FFT	Fast Fourier Transform
FIFO	First In First Out

FT	Fourier Transform
GA	Genetic Algorithm
GPIO	General Purpose Input/Output
GPU	Graphical Processing Unit
HC	Hydro Carbon
HD	High Definition
HHT	Hilbert-Huang Transform
HPF	High Pass Filter
HT	Hilbert Transform
Hz	Hertz
H/W	Hardware
ICE	Internal Combustion Engine
IEMSPCA	Industrial Extended Multi-Scale Principle Components Analysis
IMF	Intrinsic Mode Functions
IR	Inner Race
ISR	Interrupt Service Routine
I2C	Inter-Integrated Circuit
I2S	Inter-IC Sound
I/O	Input/Output
JTAG	Joint Test Action Group
KF	Kalman Filter
MISO	Master In Slave Out
MLP	Multi-Layer Perceptron
MOSI	Master Out Slave In
MSE	Mean Square Error
MSPCA	Multiscale Principal Component Analysis
NHTSA	National Highway Traffic Safety Administration
NN	Neural Network
OBD	On-Board Diagnostics
OR	Outer Race
PCA	Principle Component Analysis
PCB	Printed Circuit Board
PC	Principle Component
POSIX	Portable Operating System Interface
PWM	Pulse Width Modulation
RAM	Random-Access Memory

RBC	Reconstruction Based Charts
RF	Radio Frequency
RISC	Reduced Instruction Set Computer
RMS	Root Mean Square
RTA	Rapid Thermal Annealing
RTC	Real Time Clock
SMC	Sliding Mode Control
SNR	Signal-to-Noise Ratio
SoC	System-on-Chip
SOHC	Single Over Head Cam
SPE	Squared Prediction Error
SPI	Serial Peripheral Interface
SRAM	Shadow Random Access Memory
SSCP	Sums of Square and Cross Products
SSE	Sum Square Error
STFT	Short Time Fourier Transform
SVM	Support Vector Machines
SVSF	Smooth Variable Structure Filter
S/W	Software
LED	Light Emitting Diode
Li-Po	Lithium Polymer
LIN	Local Interconnect Network
LM	Levenberg-Marquardt
LPF	Low Pass Filter
TCP/IP	Transmission Control Protocol/Internet Protocol
TDC	Top Dead Center
UART	Universal Asynchronous Receiver/Transmitter
uC	MicroController
uP	MicroProcessor
USB	Universal Serial Bus
WPT	Wavelet Packet Transform
WT	Wavelet Transform

List of Symbols

$e_{k\|k}$	A Posteriori Estimation Error
$e_{k+1\|k}$	A Priori Estimation Error
v_k	Local Gradient of the Node k
w_{kj}	Weight From Node j to Node k
$\mathrm{X}(\omega)$	Continuous Fourier Transform
X_m	Discrete Fourier Transform
$x(n)$	Discrete Time Signal
$x(t)$	Continuous Time Signal
$X_w(a,b)$	Wavelet Transform
x_{new}	New Observation or Measurement
$\hat{x}_{k\|k}$	A Posteriori Estimation of the Weight
$\hat{x}_{k+1\|k}$	A Priori Estimation of the Weight
X	Data Matrix
y_k	Output of the Node k
$\hat{z}_{k\|k}$	A Posteriori Estimation of the Measurement
$\hat{z}_{k+1\|k}$	A Priori Estimation of the Measurement
$C_{n\|linearized}$	Jacobian Matrix of the Measurement Function a the kth Iteration
C^+	Pseudo Inverse of Matrix C
H	Hessian Matrix
I	Identity Matrix
J	Jacobian Matrix
P	Principle Component Loading Matrix
S_k	Residual Covariance Matrix at the kth Iteration

T	Principle Component Score Matrix
Λ	Eigenvalue Diagonal Matrix
Σ_T	Diagonal Matrix of Principle Components
α	Significance Level
$\mathcal{T}^2$	Hotteling's T-Squared Index
$\mathcal{Q}$	Squared Prediction Error Index
φ	Combined Index
δ	Upper Limit of SPE Index
τ	Upper Limit of Hotteling's Index
Φ	Upper Limit of the Combined Index
F_c	Fault Isolation Index
S_c	Fault Detection Index
ζ	Chi-Squared Distribution Upper Limit
g_φ	Chi-Squared Distribution Weights
h_φ	Chi-Squared Distribution Degree of Freedom
γ	SVSF Gain Parameter
C_ψ	Wavelet Admissibility Condition Parameter
$\mathcal{L}^2(\mathbb{C})$	Hilbert Space
θ	Trace of the Eigenvalue Matrix
$\psi(\cdot)$	Mother Wavelet Function
$\Psi(\cdot)$	Integral of the Wavelet Function
$\mathrm{F}_\alpha(\cdot)$	Ƒ-Distribution
$\mathcal{X}^2_\alpha(\cdot)$	Chi-Squared Distribution
$\varphi(\cdot)$	Neuron's Activation Function
$d_j(\cdot)$	Desired Output at Node *j*

$o_j(\cdot)$	Computed Output at Node j
$e_j(\cdot)$	Error Signal at Node j
$\delta_j^{(l)}(\cdot)$	Local Gradient at Node j in Layer l
$\Delta(\cdot)$	Difference Vector
$E(\cdot)$	Total Sum Square Error Signal
$g(\cdot)$	Gradient Function
$CWT_f(\cdot)$	Continuous Wavelet Transform of the Signal f
$DWT_f(\cdot)$	Discrete Wavelet Transform of the Signal f
$RBC(\cdot)$	Reconstruction Based Contribution Charts
$COV(\cdot)$	Covariance Function
$W_\phi[\cdot]$	DWT Approximation Coefficients
$W_\psi[\cdot]$	DWT Detail Coefficients
$\phi_{j,k}[\cdot]$	Scaling Function
$\psi_{j,k}[\cdot]$	Wavelet Function
$\widetilde{(\cdot)}$	Residual Subsystem
$\widehat{(\cdot)}$	Feature Subsystem
$\overline{(\cdot)}$	Averaging notation
$(\cdot)^T$	Transpose of the Matrix
$(\cdot)^{-1}$	Inverse of the Matrix
$(\cdot)'$	First Order Derivative
$\lvert\cdot\rvert_{abs}$	Absolute Value Operation
$(\circ)$	Hadamard's Entry Wise Multiplication

Table of Contents

List of Tables

List of Figures

Chapter 1. Introduction

1.1. Overview

When first invented, the early prototypes of the Internal Combustion Engine (ICE) used gun powder to drive the pistons inside the engine cylinders [1]. These first attempts paved the way to solving two main problems: finding the right ignition fuel and having controlled combustion between cycles. Eventually, the first successful gas engine was created by Etienne Lenoir, in 1859, and was refined 20 years later by Nikolaus Otto, in 1878, allowing the commercialization of this machinery [1]. In his invention, Otto adopted the four-stroke cycle, otherwise known as the Otto cycle that entails induction, compression, firing, and exhaust. In thermodynamics, this cycle translates into an isentropic compression (where entropy is constant), an isothermal heat supply, an isentropic expansion, and an isothermal heat rejection [2]. These different engine-steps produce work and power by subjecting a mass of gas to changes in pressure, temperature, volume, and heat.

In broad terms, categories of engine technologies include: spark ignition engines, diesel engines, and gas turbines [3]. While the first 2 types of engines are highly tailored towards the automotive industry, the last category is mainly used in the aerospace sector as a result of its high power-to-weight ratio. Conversely, gas turbines produce power through a continuous process of combustion, whereas both diesel and gasoline engines follow the same discretized steps of intake stroke, compression stroke, power stroke, and exhaust stroke.

For the case of the Otto cycle gasoline engine, see Figure 1.1, a new cycle starts when the piston inside the cylinder moves from the top-dead center (TDC) to the bottom-dead center (BDC) brining a mixture of air and fuel into the cylinder. This is followed by a compression step where the piston moves in the opposite direction (from BDC to TDC) which results in a hot air-fuel mixture. The third stroke begins when the spark plug induces the combustion inside the cylinder which generates

power and moves the piston down toward the BDC. Finally, the cycle ends with the exhaust stroke where the burned fumes are driven out from the cylinder by the upward motion of the piston.

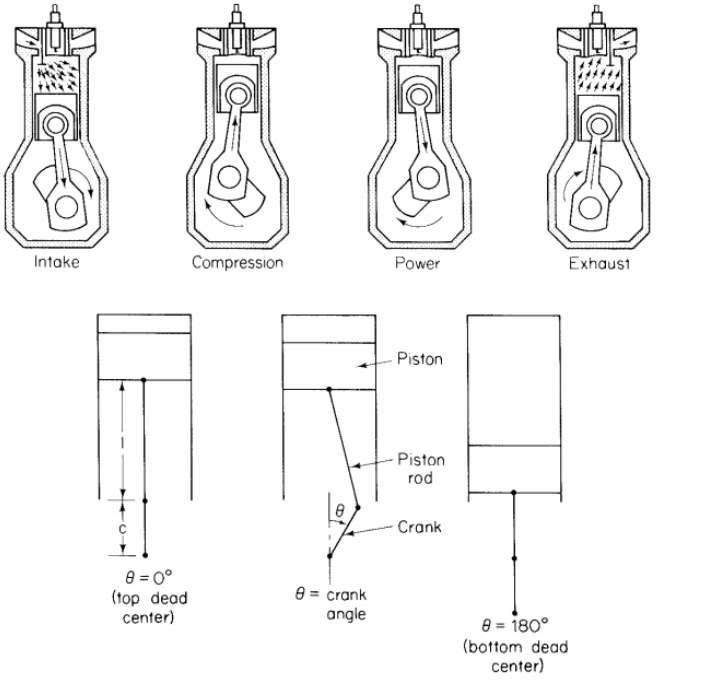


Figure 1.1 Four-Stroke Spark Ignition Engine [4]

Internal combustion engine technologies can be further subcategorised according to their principle of operation (2, 4, 6-stroke engines), their size (2.0L, 4.6L, etc.), their fuel type (gas fuel versus liquid fuel), their air-fuel mixing methods (internal versus external), their arrangement of cylinders (in-line, V, radial type), amongst others. These subcategories provide more flexibility and tailoring of the engines for their application in terms of power throughput and efficiency while adding to its complexity. Engine advancements have been enabled in parts due to the improvement of digital control technologies. The Engine Control Unit (ECU) is used to ensure optimal performance at every cycle and provide diagnostics capability. In this never-ending quest for optimal working conditions of the ICE, this research presents a functioning prototype of a fault detection and

diagnosis sensory system that monitors automatically the activities of a 4.6L Ford Single Over Head Cam (SOHC), V8, 4-stroke internal combustion engine, see Figure 1.2 and Table 1.1. The diagnostic system is mounted on a Ford Crown Victoria 2011 model, with specifications mentioned in Table 1.2. It is tailored to monitor the health of the engine in real-time in order to guarantee its optimal working performances.

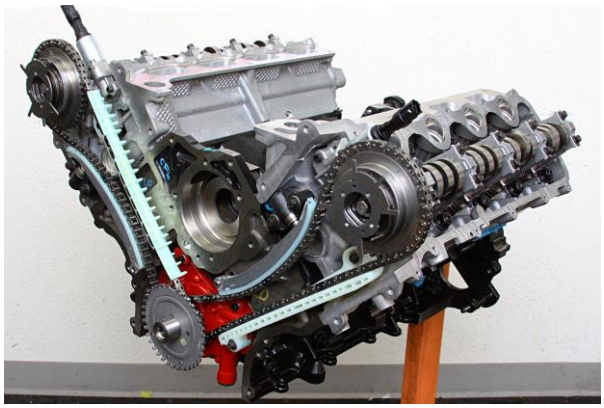


Figure 1.2 Ford 4.6L/5.4L SOHC V8 Engine [5]

ITEM	SPECIFICATION
Base Engine Size	4.60 L
Cylinders	V8
Valves	16
Cam Type	Single Over Head Cam (SOHC)
Torque	281 ft-lbs. at 4100 rpm
Horsepower	239 at 4900 rpm, 250 at 5000 RPM
Firing Order	1-3-7-2-6-5-4-8
Engine Weight	4129 lbs.

Table 1.1 Ford Crown Victoria Engine Specifications

ITEM	SPECIFICATION
Car Type	Police Interceptor Sedan
Year Model	2011
Total Seating	6
Construction	body-on-frame "Panther"
Transmission	4-speed Automatic
Engine Type	Flex-fuel (FFV) (unleaded/E85)
Drive Type	Rear Wheel drive (RWD)
Rear Axle	3:27 (129 mph) or 3:55 (119 mph)
Dimensions	212” (length), 78.3” width, 58.3” height, 114.7” wheelbase
0-60 MPH	7.61 sec
Fuel Tank Capacity	19.00 gal.
Curb Weight	4,129 lbs.
Fuel Economy (City/Highway)	16/24 mpg
Range In Miles (City/Highway)	304.0/456.0 mi.
Safety	2 front headrests, 4-wheel disc brakes ABS, Electronic brake force distribution, Engine immobilizer, Front and rear ventilated disc brakes, Tire pressure monitoring

Table 1.2 Ford Crown Victoria Specifications

1.2. Research Motivation

The latest internal combustion engines are considerably more efficient than their predecessors of the late ‘60s and early ‘70s when, emission control and fuel economy were just becoming of major concern for automotive manufacturers. Generally speaking, the amount of exhaust emissions produced by the car is tightly linked to the efficiency of its running engine; the lower the emissions, the more efficient is the engine in terms of power-to-fuel ratio.

Exhaust emission standards are continuously more stringent by forcing car manufacturers to adopt advanced closed-loop control techniques. In North America, the two main standards that have shaped the evolution of the ICE are: the Corporate Average Fuel Economy (CAFE) standards of the National Highway Traffic Safety Administration (NHTSA), and the Greenhouse Gas (GHG)

Emission Standards of the US Environment Protection Agency (EPA). As of 2016, all passenger car-manufacturers have had to comply with a CO_2 emission level of 225 g/mi and a CAFE fuel economy threshold of 37.8 mpg [6]. These limits were also applicable throughout Canada under the Canada-US Air Quality Agreement (AQA) that reduces the overlap in engine emission testing, minimizes automotive industry burden for performance monitoring, and allows for more effective use of resources (avoid duplication). To ensure compliance with these standards, all light-duty vehicles and most of heavy-duty vehicles are required to monitor selected emission system components, such as catalytic converters and fuel filler pipes, in order to notify the driver about potential inspection and maintenance checks. This vehicle's self-diagnostics status reports are accessible through the On-Board Diagnostics (OBD) port.

These standards for control and inspection technologies implicate the diagnosis and the prognostics models of the engine's condition. These models can grow in complexity as vehicles become rolling computers with ample processing power. Since the silicon chip has become very cheap, the current regular car models have on average 25 to 50 CPUs onboard. This computerization of vehicles is a *segway* to Artificial Intelligence (AI) that fuses all the information in a single brain and weighs all the acquired knowledge to make accurate prognostic decisions.

1.3. Research Objectives

This work has three main objectives:

1- Developing a Data Acquisition (DAQ) system that is powerful enough to provide high-frequency sampling of multiple sensory channels.

2- Establishing a valid FDD and prognostics that accurately combine feature extraction techniques with machine learning algorithms.

3- Comparing the implemented FDD strategies in terms of their detection, isolation, and classification capabilities.

This research is a continuation of past projects conducted at the Centre for Mechatronics and Hybrid Technologies (CMHT) of McMaster University. These include work done by:

1- Mr. Sean Hodgins (2017) [7] who developed a non-invasive wireless sensory system that is placed on the engine block to estimate its health. The setup used a wireless microcontroller unit for sampling and data transmission.

2- Mr. Feng Yifei (2016) [8] who proposed a new FDD signal processing algorithm that combines all of Crank Angle Domain (CAD) transform, different time-frequency techniques, and neural networks to detect all single-cylinder misfiring conditions with 100% accuracy.

3- Dr. Mahmoud Ismail (2015) [9] who introduced the Industrial Extended Multi-Scale Principle Components Analysis (IEMSPCA) to rapidly and accurately detect defective car alternators and starters. The proposed algorithm encompasses three principle stages. The first stage is a background noise filtration; the second stage is a combination of discrete wavelet transform and principal component analysis; and the third stage is a logistic discriminant classifier.

1.4. Contributions and Novelty

The following are the main contributions of this research:

I. The FDD data acquisition system that has a powerful microprocessor, which runs on a Linux operating system, and uses a microcontroller capable of sampling up to 8 analog channels at a high resolution of 16 bits.

II. The free reprogrammable firmware that allows the system to sample multiple sensory channels simultaneously at 40 kHz and individually at 125 kHz.

III. The Application of Feng's signal processing algorithm on knock sensor data to produce 100% accurate misfire fault detection and diagnosis.

IV. The development of a fast time-based FDD method that can detect and isolate single, dual and even triple-cylinder misfire conditions with a 100% success rate using crankshaft and camshaft only.

V. The adjustment of the extensive IEMSPCA algorithm to suit the automotive application and the addition of a neural network classifier for accurate fault classification.

VI. The recommendations of future design revisions for the next hardware prototypes and firmware updates.

1.5. Thesis Structure

This thesis has 7 chapters and they are organized as follows: chapter 2 is the literature review and covers the mathematical tools and concepts that constitute the basis of the proposed FDD algorithms. Chapter 3 considers the real-time application of the FDD algorithms to the internal combustion engine. Chapter 4 describes the data acquisition system requirement specifications to set the guidelines for building the hardware prototype. Chapter 5 illustrates in details all previous CMHT's FDD implementations. It concludes by describing the current FDD sensory system that monitors the ICE's health. Chapter 6 summarizes the experimental set-up and compares the results produced by each of the FDD algorithms. Chapter 7 reports the conclusions as well as the recommendations for future work. Appendices A, B, and C describe how to build the system, program it, and run it on the engine, and they provide all the important software codes.

Chapter 2. Literature Review

This chapter presents the literature review and covers the essential mathematical tools that make up the fault detection and diagnosis techniques considered in this work. The first section reviews the main characteristics of Fault Detection and Diagnosis (FDD) strategies. The second section covers the well-established signal processing techniques and information extraction from measured signals. An overview of neural networks is presented as well as three algorithms used in this research for their training, namely: Levenberg-Marquart (LM), Extended Kalman Filter (EKF), and Smooth Variable Structure Filter (SVSF).

2.1. Fault Detection and Diagnosis

The field of Fault Detection and Diagnosis (FDD) has been an important area of research both in industry and academia. FDD has found a wide range of applications in domains such as the process industries, microelectronics, automotive, and manufacturing, [10]. In its core, FDD aims to accomplish three main functions as follows, [11].

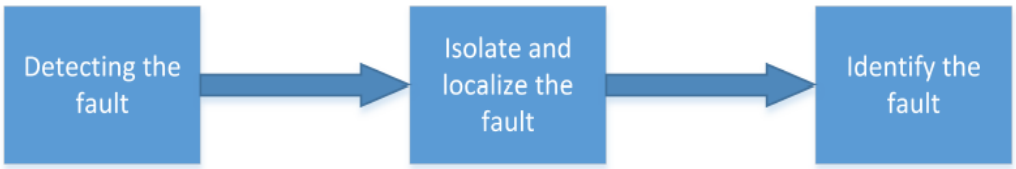


Figure 2.1 Generic FDD Algorithm Steps [9]

These functions can be automated using baseline models and intelligent algorithms to achieve Abnormal Event Management (AEM). These abnormal events, also known as faults, failures, or malfunctions, can be classified into three categories [12]:

i. Gross parameter changes in the model: refers to non-modelled parameters such as environmental parameters that disturb the process through one or more exogenous variables. When this is the case, this type of fault requires remodeling of the operations by adding the unwanted dynamics.

ii. Structural changes: refers to changes in the process itself, and usually they are closely linked to failures in the equipment, e.g. failure of a controller. When this is the case, this type of fault requires the deletion of a specific structure model as well as the update of the remaining equations to describe better the current situation of the process.

iii. Malfunctioning sensors and actuators: refers to fixed failures, e.g. permanent damage, constant biases, and out-of-range failures. This category directly influences the performance of the process' control system.

On top of these abnormal events, FDD usually operates under tight constraints related to unstructured uncertainties (non-modelled a priori faults), process noise (mismatch between the actual process and the prediction from model equations), and measurement noise (unwanted high frequency components in sensor measurements) that affect the overall performance [12]. The FDD performance is commonly assessed according to a set of characteristics such as: detection and diagnosis rates, robustness, classification, adaptability, as well as storage and computational requirements.

According to Venkakasubramanian [12], all diagnostic methods can be viewed as a specific set of transformations or mappings carried on the measured data. Looking at Figure 2.2, these sets of transformations map the measurements recorded (measurement space) into fault conditions (class space) using a three-stage process. The first mapping transforms the measured data, with no prior knowledge of the problem, into few selected features using techniques such as feature extraction

and feature selection. Reducing the data set to fewer features facilitates the decision making process later on. In the second stage, the decision space is usually obtained through a complex learning algorithm, such as neural networks or other types of classifiers. The final step involves symbolic logic discriminant (if-else conditions) accompanied by threshold functions. Across these three stages, there are two important components that constitute the foundation of every diagnostic method; these are the a priori process knowledge and the search technique. A priori process knowledge plays a crucial role in diagnostic decision making, and it highly depends on the user's understanding of the process. This knowledge can be captured in the form of invariant relationships between transducers' outputs and actuators' inputs [13] or in the form of a transformed model of a process which is summarized by a bank of filters [14] [15].

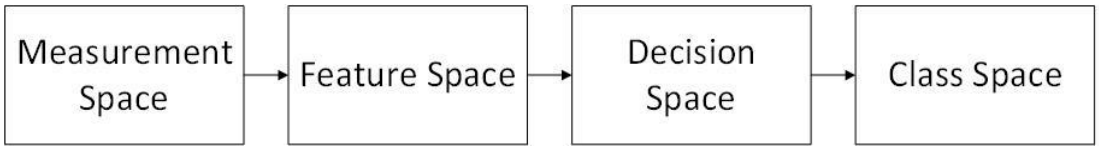


Figure 2.2 Transformations in a Diagnosis System [12]

As discussed earlier, there are two main components in designing the right FDD method: the type of knowledge that the user has about the system and the type of search in the diagnostic strategy. The latter component is usually a very strong function of the knowledge representation scheme which is a direct depiction of the a priori knowledge available [12]. Hence, due to the inherent differences in the industrial applications, there is an abundance of FDD techniques ranging from analytical methods to artificial intelligence and statistical approaches. From a modelling point of view, FDD methods can require either accurate process models, semi-quantitative models, or qualitative models. On the other end of the spectrum, there are other FDD methods that do not

assume any form of model information and rely solely on historic process data. Further classification of diagnostic approaches is summarized in Figure 2.3.

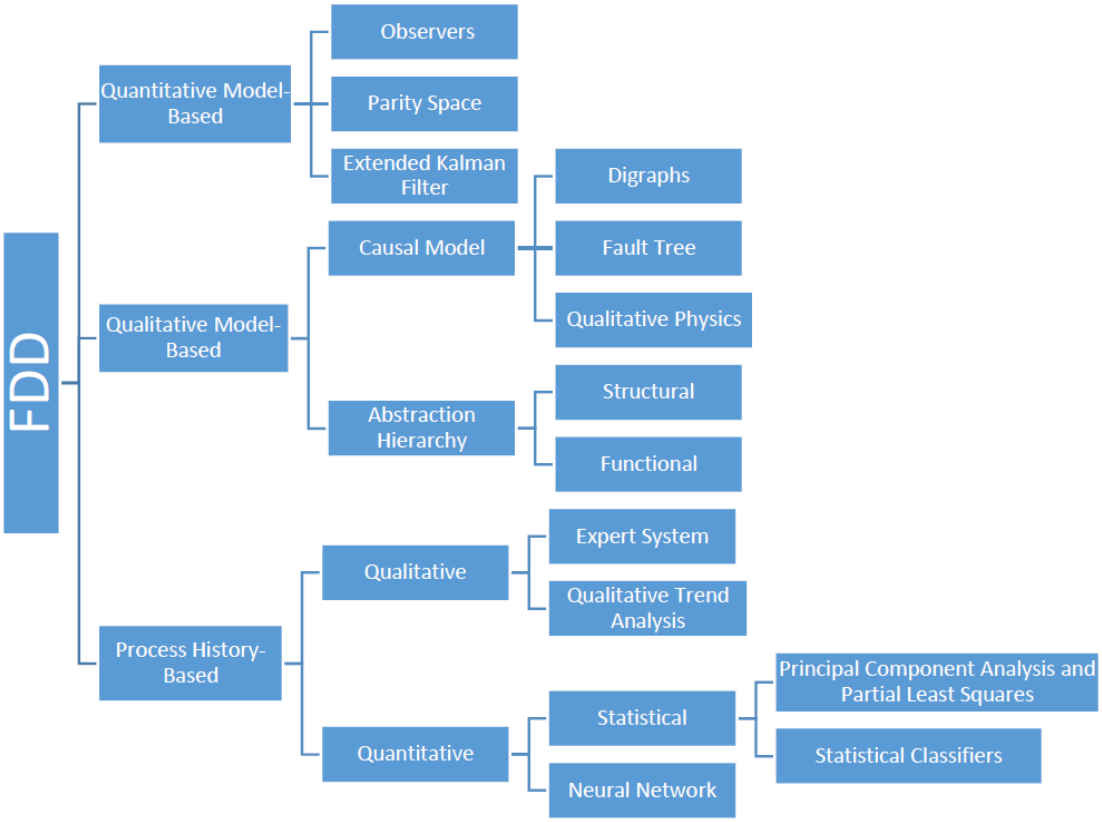


Figure 2.3 Classification of Diagnostic Algorithms [8] [12]

The model-based a priori knowledge can be broadly classified into qualitative and quantitative subclasses. Starting off with quantitative model-based approaches, the most important class of models that have been heavily investigated are input-output and state-space models [16]. Here, Kalman filters and observers are widely used for state estimation [17]. Coupled with least square methods, these tools can monitor the parameter estimates online [18]. In more recent work, more advanced techniques relying on parity equations for residual generation were developed [19]. On the other hand, qualitative model-based approaches rely heavily on the knowledge base of the

system. They are essentially a large set of *if-then-else* rules and inference engines which search through the knowledge base and derive conclusions from the given facts [20]. In such searches, what is termed as "*abductive"* reasoning is used to generate hypotheses for the source of faults resulting in an efficient bottom-up fault examination [21].

In contrast to the aforementioned model-based approaches where a priori knowledge (either qualitative or quantitative) is required, process history-based methods only need historical process data. This data provides the system with a priori knowledge about the process through means of feature extraction. The extraction can be classified further into statistical (principal component analysis and partial least squares) [22] [23] and non-statistical methods (neural networks) [24] [25].

Looking at the big picture, both qualitative and quantitative model-based methods have several desirable characteristics. In fact, when all inputs and outputs of the system, including all forms of interactions with the environment, are completely modelled, fault diagnosis becomes a well-defined problem regardless of the number of occurring faults. However, these approaches can be highly limited to the state of the sensors, their resolution, and the type of models that are used. In fact, most present models are limited to linear and some very specific non-linear systems [26]. Besides, any problem associated with the sensors used directly result in an ambiguity of the reasoning. And for that reason, these biases and drifts in the recordings have to be updated into the model in the form of uncertain parameters.

All these observations make fault detection and diagnosis applications depend heavily on the last type of methods: process history-based methods. This category of FDD techniques is easy to implement, requires very little modelling efforts and prior knowledge, performs better in terms of robustness to noise, and bypasses the need to develop dynamic models of very complicated processes, such as internal combustion engines. Nonetheless, these techniques remain limited by their generalization capabilities outside the scope of the training data. With that, any practical FDD

technique combines these complementary features to develop a hybrid method that could overcome the limitations of individual solution strategies [27]. The resulting FDD scheme has a knowledge base that combines heuristic knowledge, analytical models, input/output database, and inference engines.

In this work, since the internal combustion engine is very complex, the proposed FDD solution is a quantitative signal based method. It is a hybrid solution that combines statistical tools, e.g. PCA, with a trained model of neural network. To record improved performances, the applied method has to address the problem of optimal sensor location to enhance its observability, detectability, and data reconciliation capabilities, such as automatic detection of sensor faults and biases.

2.2. Signal Processing Techniques for FDD

This section addresses the theory and the practical-implementations of signal processing techniques that are used for feature extraction. This extraction is done in either frequency domain (Fourier transforms) or time-frequency domain (wavelet transforms). Other techniques are statistical in nature and ensure feature extraction through a defined set of analytical indices, such as Principal Component Analysis (PCA), and Multi-Scale PCA (MSPCA).

2.2.1. Fourier Transform

Fourier Transform allows spectral analysis of time-domain signals. The frequency-domain representation, also known as spectrum of the signal, helps to detect and isolate any recurrent events in a signal. Nowadays, Fourier transformation is a fundamental tool used across different applications stretching from image compression [28] to speech recognition [29] [30].

Fourier analysis techniques are broadly categorized as Fourier transforms and Fourier series and, have been derived for both continuous as well as discrete time analysis. While the Fourier series

apply to periodic signals only, Fourier transforms apply to any signal. The majority of application involve sampled data and signals and rely on the discrete time formulation of the Fourier transforms (DFT) and its more computationally efficient form of Fast Fourier Transform (FFT) [31] [32].

The ability of Fourier transforms to extract and isolate the signal component frequencies associated with fault conditions makes this tool very useful. Chinmaya et al. [33] proposed a method that relies on a multiresolution Fourier transform to carry fault diagnosis in a multistage gearbox which was subject to transient loads. The signal-based FDD scheme used vibration transients, which were recorded from an accelerometer located at the tail-end of the gearbox, and current transients from a motor. The vibration data demonstrated that the gearbox load removal had negligible effect on the frequency content of this signal. Meanwhile, measured current proved that gearbox defects result in transients that smear the spectral interpretation of the FFT results. In this application the Fourier transform served to highlight the high-frequency components resulting from faults in both vibration and current signals and, enabled the development of a fault-monitoring scheme for a gearbox with varying load conditions.

Rai et al. [34] combined FFT results with Intrinsic Mode Functions (IMFs) which are derived from Hilbert-Huang Transform (HHT) to maximize the efficiency of Hilbert Transform (HT) spectrum analysis related to bearing faults. As these bearing faults constitute the most common cause of machinery breakdown, it was mandatory to develop a robust technique to detect any incipient defects. The developed method was able to clearly detect and isolate all bearings' defect frequencies up to the fourth harmonic. Besides capturing these faults accurately, the additional FFT-step demonstrated the presence of an amplitude modulation between Inner Race (IR) and Outer Race (OR) faults which allowed to better the performance of the model-based FDD approach.

Wu et al. [35] used FFT to establish *vibro-acoustic* predictions to discern the mechanism of claw pole alternators. The research was dedicated to reducing the noise level along with the

electromagnetic vibrations present by this common type of alternators. To achieve that, FFT was used to generate and analyse the spectrum of nodal forces felt by the stator's inner surface. Furthermore, this technique allowed to compare the different spectrums produced between the measured and the simulated vibration and noise results. In doing so, the numerical benefits of FFT contributed to the creation of an accurate model of a three-phase 12-pole/36-slot claw pole alternator. This model showed that the electromagnetic noise and vibration are mainly caused by the zeroth-order force (circumferential spatial-order) whose harmonics occupy specific set of spatial-frequency bins. Additionally, the created model proved the strong correlation between the magnetic forces (not the torque ripples) and electromagnetic vibration/noise in such type of alternators. By that, all these findings improved the installation conditions of both stator core and armature windings. This resulted in a higher efficiency and a better output performance of modern automobiles' power generators (alternators).

2.2.2. Wavelet Transform

As mentioned in the previous paragraph, the Fourier transform is an effective mathematical tool to highlight frequency components of a certain process. Although this transform provides the frequency contents of a particular fault, it lacks the capability of pinpointing the specific time when these frequency contents emerge. The Wavelet transform, on the other side, overcomes this problem by allowing the analysis of the spectrum in both time and frequency domains simultaneously. This enables the development of FDD algorithms that are applicable to transient, non-stationary, and time-varying phenomena. Initially, the driving equation of wavelet transform was proposed by the mathematician Alfrd Haar in 1909. The concept of wavelets was later introduced by the geologist Jean Morlet who coined the term 'wavelet' in 1984 [36]. By definition, a wavelet is a small wave that has an oscillating wavelike characteristic and an energy concentrated around a specific time. By that, wavelets are waveforms that are limited in duration and have an

average value of zero. Figure 2.4 draws the differences that exist between a smooth, predictable, and everlasting wave (sine wave) and a limited, irregular, and sometimes asymmetric wavelet.

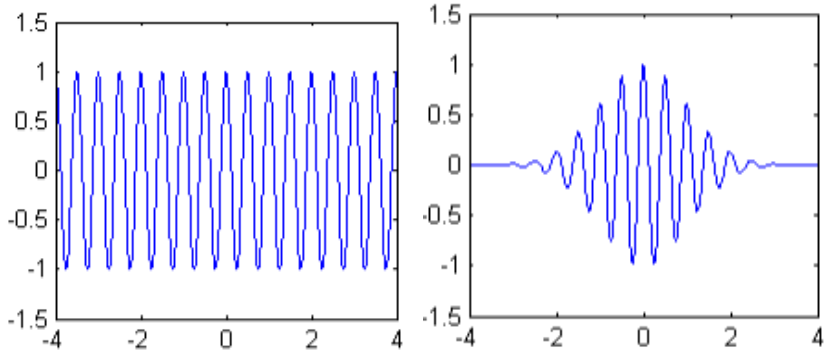


Figure 2.4 Wave (left) Versus Wavelet (right) [37]

Wavelet transforms are the inner product of the signal and a specific family of the wavelet, see equation (2.1). For this equation, $\psi(t)$ constitute the mother wavelet which represent the prototype function of a family of wavelets. The other elements within the same family are a series of children wavelets $\psi_{a,b}(t)$ that are generated by dilation and translation from the mother wavelet $\psi(t)$, see equation (2.2).

$$X_w(a,b) = \frac{1}{|a|^{1/2}} \int_{-\infty}^{\infty} x(t)\psi\left(\frac{t-b}{a}\right) dt \,, a \,\epsilon \mathbb{R}^{+} \, and \, b \,\epsilon \mathbb{R} \tag{2.1}$$

Where

$$\psi_{a,b}(t) = \frac{1}{|a|^{1/2}} \psi\left(\frac{t-b}{a}\right) \tag{2.2}$$

In the above two equations the variable a stands for the scale factor, and it directly reflects the frequency contents of the wavelet. Meanwhile, the variable b symbolizes the shift factor that

mirrors the time component of the wavelet. Finally, the factor $\frac{1}{|a|^{1/2}}$ ensures energy preservation of the final transform results.

As described earlier, the prototype function $\psi(t)$ represent the mother wavelet and in order to be one, it has to satisfy the admissibility condition:

$$C_\psi = \int_{-\infty}^{\infty} \frac{|\psi(\omega)|}{\omega} d\omega < \infty, where\ \psi(\omega) = Fourier\ Transform(\psi(t)) \qquad (2.3)$$

In practice, $\psi(\omega)$ has sufficient decay that makes the wavelet act as a band-pass filter. When this is the case, the above condition (2.3) reduces to:

$$\int_{-\infty}^{\infty} \psi(t)dt = \Psi(0) = 0 \qquad (2.4)$$

Wavelet transform has two main categories: Continuous Wavelet Transform (CWT) and Discrete Wavelet Transform (DWT). CWT driving pair of equations is defined as the inner product in the Hilbert space of the $\mathcal{L}^2(\mathbb{C})$ norm, and they are as follows:

$$CWT_f(a,b) = \int_{-\infty}^{\infty} \psi^*_{a,b}(t) f(t) dt = < \psi_{a,b}(t), f(t) > \quad \text{(CWT)} \qquad (2.5)$$

$$f(t) = \frac{1}{C_\psi} \iint_{-\infty}^{\infty} CWT_f(a,b)\, \psi_{a,b}(t) \frac{da\, db}{a^2} \qquad \text{(Inverse CWT)} \qquad (2.6)$$

In the above two equations (2.5) and (2.6), the basis function $\psi_{a,b}(t)$, also known as the child of the mother wavelet $\psi(t)$, can also be seen as a filter bank of impulse response. In fact, as the scale factor a increases in value, $\psi_{a,b}(t)$ gets dilated in time to focus on long-time characteristics of the process' signal $f(t)$. In other words, a very large scale factor of the child wavelet means a global

view of the signal, while a very small scale factor means a detailed view of the same signal. This scale factor constitutes the resolution of the wavelet transform and it is limited by the frequency content of $f(t)$.

Similar to continuous time Fourier transforms, CWT is redundant and impractical with digital computers, and it often requires longer computing times. In practice, the parameters a and b can not take continuous values. They have to be evaluated on a discrete grid of time-scale plane ($a = a_0^j,\ b = k a_0^j b_0, where\ j,k\ \epsilon$ Z) leading to a discrete set of children wavelet functions described by:

$$\psi_{j,k}(t) = a_0^{-j/2}\psi\left(a_0^{-j}\, t - k\, b_0\right) \tag{2.7}$$

These discrete children wavelets constitute the basis for the discrete wavelet transform pair:

$$DWT_f(j,k) = \int_{-\infty}^{\infty} \psi_{j,k}^*(t) f(t) dt = < \psi_{j,k}(t), f(t) > \text{ (DWT)} \tag{2.8}$$

$$f(t) = \sum_j \sum_k DWT_f(j,k)\, \psi_{j,k}(t) \qquad \text{(Inverse DWT)} \tag{2.9}$$

The above equation (2.9) can be approximated for a given scaling function ϕ and a wavelet function ψ. The reconstructed discrete signal $f(n)$ can be approximated as such:

$$f(n) = \frac{1}{\sqrt{M}} \sum_K W_\phi[j_0,k]\phi_{j0,k}[n] + \frac{1}{\sqrt{M}} \sum_{j=j0}^{\infty} \sum_K W_\psi[j,k]\psi_{j,k}[n] \tag{2.10}$$

Where

$$\phi_{j,k}[n] = 2^{j/2}\phi\left(2^j n - k\right) and\ \psi_{j,k}[n] = 2^{j/2}\psi\left(2^j n - k\right) \tag{2.11}$$

For both equations (2.10) and (2.11), M represent the number of samples of the discrete signal. Accordingly, both $\phi_{j,k}[n]$ and $\psi_{j,k}[n]$ are defined only in the interval [0, M-1]. Finally, both $j\ and\ k$ are respectively the scaling and the shift parameters of the discrete wavelet transform. To obtain the approximate DWT coefficients, $W_\phi\ and\ W_\psi$, from the original signal $f(n)$, the following two equations are applied:

$$W_\phi[j_0,k] = \frac{1}{\sqrt{M}}\sum_n f(n)\phi_{j0,k}[n] \quad \text{(DWT approximation Coefficients)} \qquad (2.12)$$

$$W_\psi[j,k] = \frac{1}{\sqrt{M}}\sum_n f(n)\psi_{j,k}[n]\,, j \geq j_0 \qquad \text{(DWT detail Coefficients)} \qquad (2.13)$$

These aforementioned equations pave the way to an analogous technique to FFT known as fast wavelet transform [38]. Wavelet coefficient-approximation is explained by Figure 2.5 and the reconstruction of the original signal is summarized in Figure 2.6. For both figures, $G_0(z)$ & $H_0(z)$ are low-pass filters, $G_1(z)$ & $H_1(z)$ are high-pass filters, $A_{j,j=1..L}$ are the approximation coefficients, $D_{j,j=1..L}$ are the detail coefficients, and L is the level of the DWT. The forward pass of Fast DWT produces the wavelet coefficients through a sequence of filtering and down-sampling steps applied on the original signal $f(n)$. Meanwhile, the Inverse Fast DWT takes the produced wavelet coefficients and applies a train of up-sampling and filtering steps until the original signal $f(n)$ is recovered.

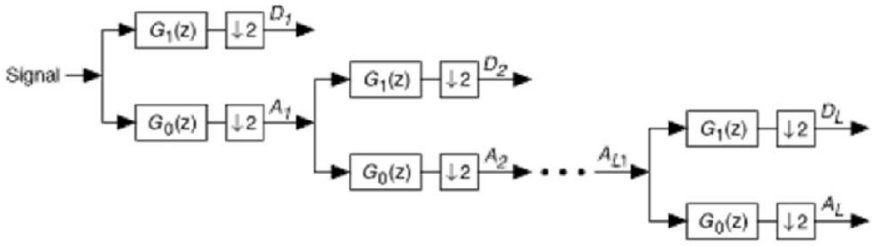


Figure 2.5 Schematic Diagram for Fast Wavelet Transform [38]

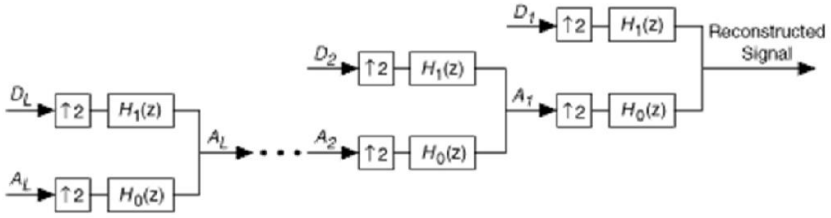


Figure 2.6 Schematic Diagram for Inverse Fast Wavelet Transform [38]

Wavelet functions, either discrete or continuous, can take any form as long as the admissibility condition, described by equation (2.3), is satisfied. All order and scaling variations of the same function are grouped into classes of wavelet families. Based on their common properties, e.g. advantages and disadvantages, these wavelet families can be clustered into 5 main groups. Table 2.1 illustrates the aforementioned points in details. For a given task, it is challenging to select the most optimum mother wavelet. Added to that, different selections of the mother wavelet lead to completely different results. Besides, considering that many wavelet families share the same properties, there is no fixed standard yet that defines the selection process. However, generally speaking, to decide on a particular mother wavelet, properties, such as orthogonality, compact

support, symmetry, and vanishing moment, represent a good selection reference. As a rule of thumb, this selection is based on the shared energy and entropy with the original signal. While the energy reflects the similarity points between the wavelet and the signal, entropy mirrors the amount of data missing between the signal and mother wavelet. A selected mother wavelet should have a high energy and a low entropy. Following on this, Table 2.2 presents the shape of some of the most common wavelet family functions.

Group Name	Wavelet examples	Advantages	Disadvantages
Crude Wavelets	Gaussian, Morlet, Mexican hat	Symmetry, explicit expression available	Fast algorithms + reconstruction are unavailable
Infinitely Regular Wavelets	Meyer, Discrete Meyer	Symmetry, infinite regularity	Fast algorithm is unavailable
Orthogonal and compactly supported wavelets	Daubechies, Symlets, Coiflets	Compact Support, vanishing moment, FIR filters	Poor regularity
Biorthogonal and compactly supported wavelet pairs	B-splines biorthogonal wavelets	Symmetry with FIR filters	Orthogonality is lost
Complex wavelets	Complex Gaussian, Complex Monet, Complex Shannon	Symmetry, explicit expression available	Fast algorithms + reconstruction are unavailable

Table 2.1 Different Wavelet Families [9]

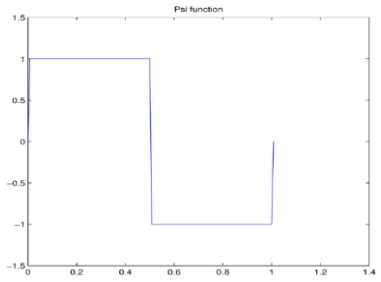
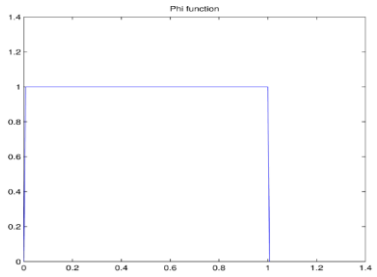
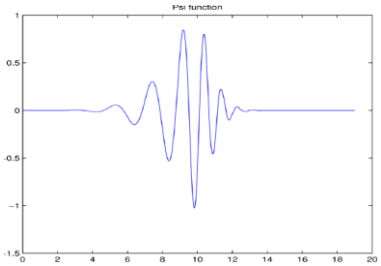
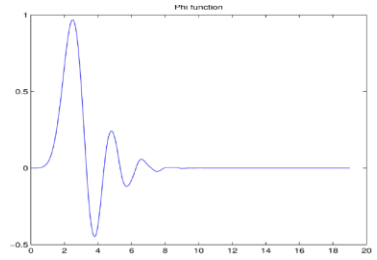
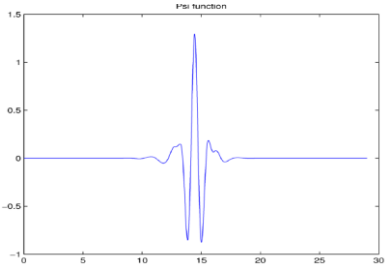
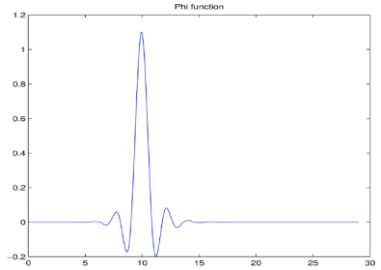
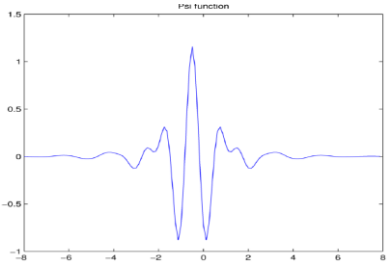
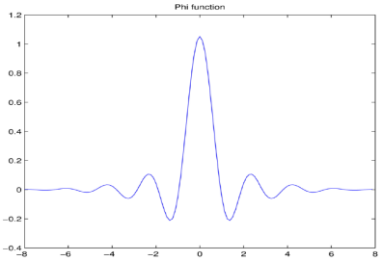
Family Name	Order	Wavelet Function	Scaling Function
Haar	1		
Daubechies	10		
Coiflets	5		
Meyr	-		

Table 2.2 Few Examples of Wavelet Family Functions [9]

Wavelet transform is a very powerful tool in FDD applications. Wu and Liu [39] used an orthogonal type of discrete wavelet transform to extract the features from an internal combustion engine's sound emission signals. Fault diagnosis was carried using three different scaling orders of Daubechies, 'db4','db6', and 'db20'. These scaling orders helped differentiate between five synthetic faults, including air leaking of intake manifold, Electronic Control Thermal (ECT) sensor fault, cam-shaft sensor fault, one-cylinder miss-firing , and two-cylinder miss-firing that were recorded at different engine speeds of 750 rpm, 1000 rpm, 2000 rpm, and 3000 rpm. Combined with neural networks, the extracted features were all classified with a performance over 95%. This work has also shown that 'db20' was particularly more accurate than the lower order Daubechies, notably 'db4' and 'db6'. To sum up, the combined solution of DWT and NN techniques was able to detect and classify several faults of a V-type, 6-cylinder internal combustion engine.

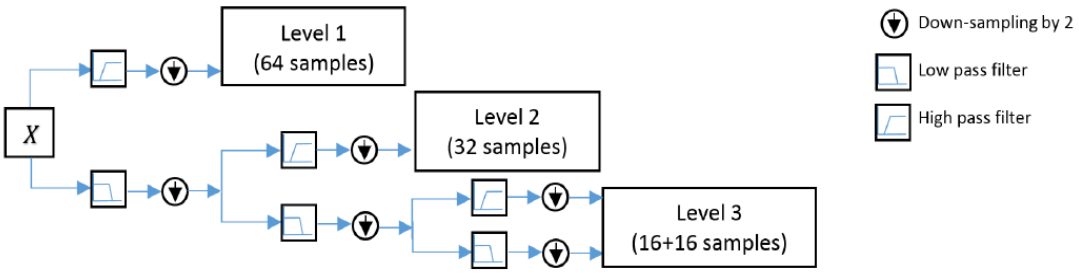
Kozionov et al. [40] used wavelet transform as a viable validation method to differentiate between sensor faults and system dynamics. The proposed solution logged multiple temperature sensors scattered across a gas turbine to create a two-step multidimensional validation scheme. In the first step, wavelet transform was applied to detect the signal changes both at high and low frequencies. The produced results were then inspected using dynamic methods, e.g. universal threshold method and median threshold subtraction. The final multi-dimensional method flagged all sensor faults and identified all gas turbine dynamic stages despite the incurrent harsh conditions.

Lin and Qu [41] used the Morlet wavelet family as an effective feature-extraction method for processes that have low signal-to-noise ratio (SNR). The Morlet showed to be a very effective de-noising tool when carrying impulse component extraction of mechanisms that operate in loud industrial environments. To test these assumptions, the researchers applied this FDD scheme on rolling bearings. There, defects often produce vibrational impulses as rollers pass through a crack in either Outer Race (OR) or Inner Race (IR) of the bearing. And by using Morlet wavelets

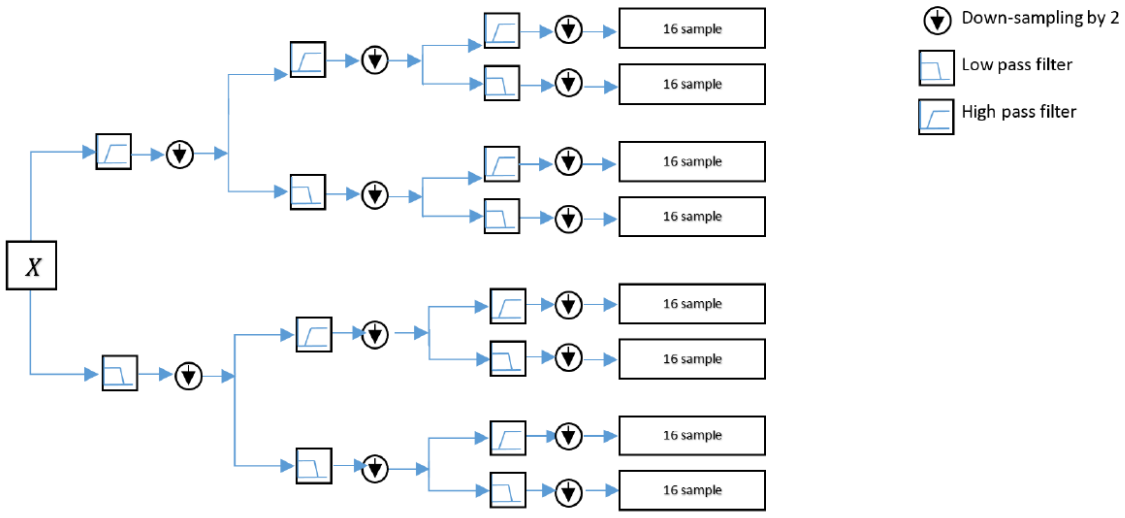
researchers were able to pick up early symptoms of gearboxes' tooth damage which can prevent related car accidents. Therefore, like with the aforementioned cases, wavelet transform is undeniably a required step when developing FDD schemes for internal combustion engines.

2.2.3. Wavelet Packet Transform (WPT)

The Discrete Wavelet Transform (DWT) highlights mainly the high-frequency components. This transform is different from the Wavelet Packet Transform (WPT) that divides the frequency spectrum equally. Considering that there is no prior knowledge of the system's fault characteristics, this second approach presents a better mathematical tool for FDD applications as faults can manifest both in high and low frequencies with equal probabilities. Figure 2.10 shows the difference between DWT and WPT where WPT solves the frequency-band disagreement produced by the DWT.



(a) Discrete Wavelet Transform (DWT)



(b) Wavelet Packet Transform (WPT)

Figure 2.7 Comparison Between WPT and DWT [9]

In calculating the next level approximation and detail coefficients of the WPT, the following two equations are used:

$$W_{j+1,2k}[n] = W_{j,k} * g_j[2n] \tag{2.14}$$

$$W_{j+1,2k+1}[n] = W_{j,k} * h_j[2n] \tag{2.15}$$

Where $W_{j,k}$ are the wavelet coefficients at the level j for the frequency bin k. g_j *and* h_j are respectively high-pass and low-pass filters coupled with a decimation of order 2. Depending on the parity of the index k, the computed coefficients result in a low-pass filter (k even) or higher-pass filter (k odd). At the last level j, the above two equation (2.14) and (2.15) distribute the frequency bandwidth equally for all coefficients ($W_{j,k}$). Figure 2.8 provides an example of a 3-level WPT for an original signal with a sampling frequency F_s. The WPT is applied on the Nyquist sampling

frequency range of $F_s/_2$. At the first level, this Nyquist frequency is divided into high-frequency contents ($W_{1,0}$) and low-frequency contents ($W_{1,1}$) with equal size $\Delta_f = \left.\frac{F_s/_2}{2^j}\right|_{j=1} = \frac{F_s}{4}$. This decomposition repeats until the 3rd level is reached where the WPT coefficient ($W_{3,k}, k = 1..7$) represent a frequency ranges that are equal to $\Delta_f = \left.\frac{F_s/_2}{2^j}\right|_{j=3} = \frac{F_s}{16}$ in size. This balanced division of the original frequency interval makes WPT the superior time-frequency analysis for nonstationary signals with middle and high frequency contents

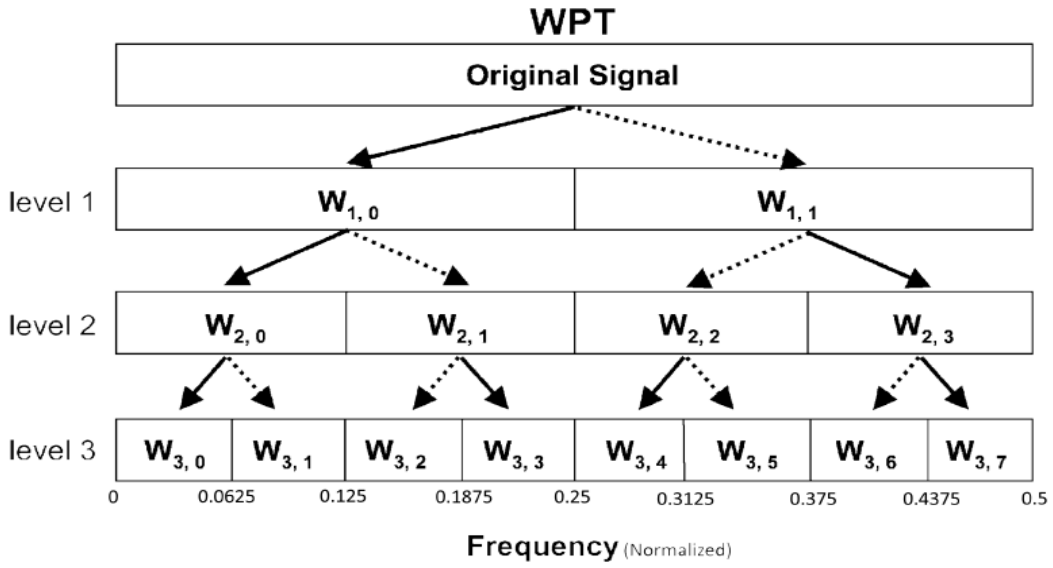


Figure 2.8 Frequency Content of WPT Coefficients [9]

Wavelet Packet Transform is a viable tool for Fault Detection and Diagnosis. Wu et Liu [42] employed WPT with Shannon entropy in order to carry the feature extraction of fault conditions from an engine's sound recordings. A microphone was connected to the National Instruments (NI-6024E) data acquisition system and logged the engine activities under different operating conditions: healthy, air leakage of the intake manifold, camshaft sensor fault, electronic control

thermal (ECT) sensor fault, and misfiring faults. The engine was the Mitsubishi V type, four-stroke, six cylinder, and 3.0L Gasoline Direct-Injection (GDI), and it operated at three speeds: 750 rpm (idle), 2000 rpm, and 3500 rpm. The results produced by the WPT trained two different networks: the conventional Back-Propagation Network (BPN) and the Generalized Regression Neural Network (GRNN). The classification results were compared between the two topologies which produced an overall accuracy above 95%.

Wang et al. [43] proposed the Waveform Feature Manifold (WFM) method which is based on the Wavelet Packet Transform (WPT) to extract weak bearing faults signals. The targeted faults are known for having a weak transient signals with wide spread frequency characteristics and a low signal-to-noise ratio as shown by the red waveform in Figure 2.9. The diagnostics capabilities of the WFM was compared to kurtogram-based methods which computes the kurtosis to extract the presence of transient impulse responses. To test the validity of the new method, two experimental fault conditions were considered: weak outer race defects, and weak rolling element defects. The bearing speed was kept fixed at 2000 rpm. The accelerometers were mounted on the housing of each bearing and were sampled at 20 kHz. The results of the experiment demonstrated that WFM, in particular WPT, outperformed the kurtogram-based methods to effectively extract the bearing faults signatures from weak signals. This makes the WFM technique reliable for flagging early stage faults before they lead to fatal breakdowns in rotating machinery.

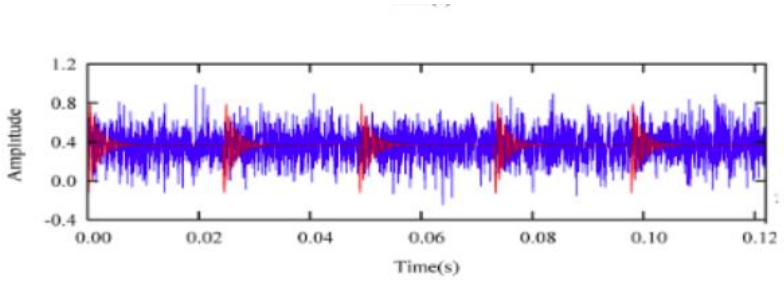


Figure 2.9 Simulated Weak Bearing Transient Impulses (red)

Zarei and Poshtan [44] used the Wavelet Packet Transform (WPT) on the induction motor's stator current signal to detect bearing faults. Monitoring the stator current provided a non-intrusive approach, compared to vibration signal analysis, in order to localize common type of bearing defects, namely outer race, inner race, and distributed defects. Meyer was selected as the mother wavelet for the WPT. The diagnostics technique started by using WPT to decompose the stator current which was measured under different conditions. The WPT isolated the defect frequency range which allowed to calculate the coefficient energies on the zoomed defect frequencies. The obtained results were compared against the healthy measurement to produce fault indices for a viable diagnostics. To test the WPT-based FDD technique, a three-phase, 1.2 KW, 380V, 50 Hz, 1400 rpm, four pole induction motor was used. 5 holes with different thicknesses were drilled both on the inner and outer race of the motor's bearing. WPT demonstrated great capabilities to detect the lightest defects in the bearings (smallest drilled holes). Therefore, the stator current analysis combined with WPT represent a reliable alternative to vibration-based fault detection and diagnosis techniques.

2.2.4. Principal Component Analysis (PCA)

Principal component analysis' origins can be traced back to 1829 with Cauchy [45] and later with Pearson [46] in 1901. However, its name was first coined by Hostling [47] in 1933. PCA is a multivariate statistical technique that uses orthogonal transformation to convert a set of values of potentially correlated variables into a set of values of linearly uncorrelated variables (smaller in size), called 'Principal Components' (PC). PCA reduces the multidimensional data into lower dimensions with negligible information loss, and it is relevant to applications such as dimension vectors reduction for face recognition and image compression.

In most cases, when examining the directions with the largest variations, PCA is performed on a square symmetric matrix representation of the data set. The most common types of such matrices

are: SSCP matrix (pure sums of square and cross products), covariance matrix (scaled sums of square and cross products), and correlation matrix (sums of squares and cross). By transforming the data set, PCA can manage data without considering its fundamental class structure (no bias), and it remains computationally efficient even when the data set is big. Nevertheless, PCA suffers from accuracy issues especially when trying to evaluate either of the aforementioned square matrices.

According to Abdi et al. [48], PCA has mainly 4 goals: extract the most important information from the data set; compress the size of the data set; simplify the description of the data set; and analyze the structure of the observations and variables from the data set. And to achieve these goals, PCA computes the principal components, which are a linear combination of the original variables, as follows: the first principal component is calculated to have the largest possible variance. Similarly, the second component is computed with the same principle but under the constraint of being orthogonal to the first component. Similarly, the remaining PCs can be generated according to this rule. In order to illustrate this point better, let x_1 and x_2 be the input variables and t_1 and t_2 the output variables of the PCA. As shown in Figure 2.10, both principal components (t_1 and t_2) are orthogonal to each other. On top of that, the axis along t_1 has a higher variance than that of the axis lead by t_2. This means that the information contents residing in t_1 are considerable compared to those prescribed by t_2.

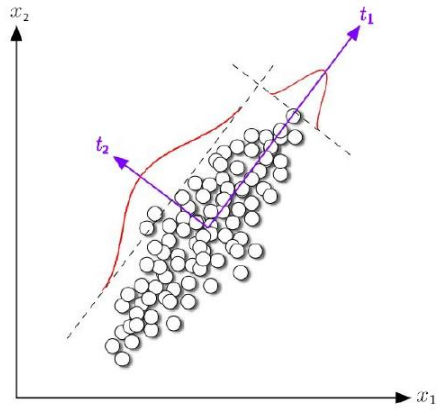


Figure 2.10 Principal Component Analysis [49]

The main driving equation for Principal Component Analysis is as follows:

$$T = XP \tag{2.16}$$

Where X represent the original $n \times m$ data-matrix that contains n measurements of m variables. Furthermore, P and T are respectively called the *principal components loading* and the *principal component scores*. While the columns of the $m \times m$ square-matrix P represent the basis vectors of the new principal component space, the columns of the $n \times m$ T matrix are the resulting uncorrelated signals of the cross-correlated data which was extracted from the matrix X.

PCA is a great candidate for fault detection and diagnosis, as it is capable of exposing faulty recordings that usually manifest by an increase in variance from the normal healthy signals. To demonstrate this, according to equation

(2.17) [50], the data matrix X can be decomposed into two orthogonal subspaces. The first subspace $\hat{X}$ contains the first l principal components that accounts for the major portion of the variance in the system. The second subspace is made up of the $(m-l)$ residual principal components that mirror the amount of noise and error in the faulty signal.

$$\begin{aligned} X &= \hat{X} + \tilde{X} \\ &= \hat{T}\hat{P} + \tilde{T}\tilde{P} \\ &= \begin{bmatrix} \hat{T} & \tilde{T} \end{bmatrix} \begin{bmatrix} \hat{P} & \tilde{P} \end{bmatrix}^T \\ &= TP^T \end{aligned} \tag{2.17}$$

In the above equation, the principal component subspace $\hat{X}$ is the approximated (filtered) version of the original data matrix X. In order to make such approximation, there are several threshold techniques developed in the literature. Two of the most common ones are: Heuristic rule and Kaiser's rule. While the first rule selects the main principle components so that 95% of the total variance is kept, the second technique selects all PCs with a higher variance than the mean computed variance.

Once the above decomposition of the data matrix is accomplished, all recorded PCs get translated into appropriate statistical indices in order to carry FDD using PCA. Joe Qin [50] summarized three main indices: Hotteling's $\mathcal{T}^2$, Squared Prediction Error (SPE) $\mathcal{Q}$, and the combination of the two previous indices in φ. All these indices provide the baseline for comparing a new observation data, x_{new}, with respect to the previously recorded healthy data.

The first index, Hotteling's $\mathcal{T}^2$, measures the variation in the PCs' space that exists between the new recording and the baseline measurements. This index picks up easily any fault that cannot

preserve the covariance relationship with the baseline measurements. And it is described by the following equation:

$$\mathcal{T}^2 = x_{new}\widehat{D}x_{new}^T , \quad \text{where } \widehat{D} = \hat{P}\widehat{\Lambda}^{-1}\hat{P}^T \text{ and } \widehat{\Lambda} = \frac{1}{n-1}\hat{T}\hat{T}^T \tag{2.18}$$

In the above equation (2.18), x_{new} is a *(1 x m)* vector containing the new observation recording, $\widehat{D}$ is positive semidefinite matrix, and $\widehat{\Lambda}$ reflects the principal eigenvalues of the filtered training data contained in $\hat{T}\hat{T}^T$ product. In this respect, the definition of $\mathcal{T}^2$ index is very suitable for real-time applications where every new data is directly fed to the trained model while updating this last at the same time.

The second index, $\mathcal{Q} - statistics$ or Squared Prediction Error (SPE), records the variations in the residual space of the residual data represented by $\tilde{X}$. It is defined as follows:

$$\mathcal{Q} = \|\tilde{x}_{new}\|^2 = \left\|x_{new}\tilde{C}\right\|^2, where\ \tilde{C} = \tilde{P}\tilde{P}^T \tag{2.19}$$

In the above equation (2.19), $\tilde{C}$ is the residual subspace where every new recording, x_{new}, is projected onto this space in order to log the variations with respect to previously monitored noise and errors in the process. With that, $\mathcal{Q} - statistics$ directly reflect all unstructured elements that the in-control PCA model cannot account for.

To produce more reliable FDD schemes using PCA, Henry and Qin [51] proposed a new global index, $\varphi - statistics$, which is a uniform combination of the 2 aforementioned indices.

$$\varphi = \frac{\mathcal{Q}}{\delta^2} + \frac{\mathcal{T}^2}{\tau^2}, where\ \delta = upper\ limit\ (\mathcal{Q})\ and\ \tau = upper\ limit(\mathcal{T}^2) \tag{2.20}$$

Or

$$\varphi = x_{new}\Phi x_{new}^{T}\,,\ where\ \Phi = \frac{\tilde{C}}{\delta_{\alpha}^{2}} + \frac{\hat{D}}{\tau_{\alpha}^{2}} \tag{2.21}$$

In the above two equations (2.20) and (2.21), the joined index φ balances both $\mathcal{Q}\ and\ \mathcal{T}^2$ with respect to each other after normalizing their contributions using their upper limits. To be able to estimate these upper limits, many researchers developed different models for $\mathcal{T}^2$. For example, Jackson [52] stated that $\mathcal{T}^2$ is tightly linked to Ƒ-distribution which allows to estimate the upper control limits of this index as follows:

$$\tau^2 = \tau_{\alpha}^2 = \frac{l(n-1)}{n-l}\mathrm{F}_{\alpha}(l, n-l), where\ \forall \mathcal{T}^2,\ \mathcal{T}^2 \leq \tau^2 \tag{2.22}$$

In equation (2.22), α is the pre-assigned significance level of the statistical distribution, l and $n-l$ are the degrees of freedom that define the Ƒ-distribution where l represent the number of representative principal components PCs and n the number of samples in the original data matrix X. On the other hand, when it comes to the second statistical index $\mathcal{Q}$, Box et al. [53] showed that the upper limits of any quadratic form, in this case $\mathcal{Q} = \left\|x_{new}\tilde{C}\right\|^2$, can be approximated by a weighted chi-squared distribution. With that, $\mathcal{Q}$'s upper limit is expressed as follows:

$$\delta = \delta_{\mathcal{Q},\alpha} = g\mathcal{X}_{\alpha}^{2}(h), \quad where\ g = \frac{v}{2m}, h = \frac{2m^2}{v}, and\ \forall \mathcal{Q}, \mathcal{Q} \leq \delta^2 \tag{2.23}$$

In equation (2.23), g and h are respectively the weight and the degree of freedom for the chi-squared distribution ($\mathcal{X}_{\alpha}^{2}$). Meanwhile, v and m are respectively the sample variance and the sample mean. Any transgression of these two aforementioned limits, $\tau^2\ and\ \delta^2$, signals the presence of faults in the system. Since both $\mathcal{Q}$ and $\mathcal{T}^2$ work in a complementary manner, there is a need to define a third threshold of their combined index φ for flagging faults in the system. Doing

so reduces the number of false negatives when carrying FDD prognostics. And in this respect, Yue and Qin [51] showed that φ −index, like Q −index, can be approximated using chi-squared distribution, and the established upper limit is defined as follows.

$$\zeta^2 = \zeta_\alpha^2 = g_\varphi \mathcal{X}_\alpha^2 \left(h_\varphi\right), \text{where } \forall \varphi, \varphi \leq \zeta^2 \tag{2.24}$$

where

$$g_\varphi = \left(\frac{l}{\tau_\alpha^4} + \frac{\theta_2}{\delta_\alpha^2}\right) / \left(\frac{l}{\tau_\alpha^2} + \frac{\theta_1}{\delta_\alpha^2}\right) \ (distribution\ weight) \tag{2.25}$$

$$h_\varphi = \left(\frac{l}{\tau_\alpha^2} + \frac{\theta_1}{\delta_\alpha^2}\right)^2 / \left(\frac{l}{\tau_\alpha^4} + \frac{\theta_2}{\delta_\alpha^4}\right) \ (distribution\ degree\ of\ freedom) \tag{2.26}$$

and

$$\theta_1 = trace\left(\widetilde{\Lambda}\right), \theta_1 = trace\left(\widetilde{\Lambda}^2\right), and\ \widetilde{\Lambda} = \frac{1}{n-1}\tilde{T}\tilde{T}^T \tag{2.27}$$

By generating all of $\mathcal{T}^2$, Q, and φ and their respective control limits τ^2, δ^2, and ζ^2, PCA can successfully carry fault detection and diagnosis. Yue and Qin [51] were able to detect and reconstruct the faults along a given fault direction using a modified version of the above φ statistics. In this new definition, both $\mathcal{T}^2\ and\ Q$ were normalized using 2 dynamic control-limits weights. By doing so, the researchers were able to approximate the fault estimation task to solving a typical optimization problem, also known as Quadratic Constrained Quadratic Programming (QCQP) problem. This approach carried accurate fault identification of 5 different types of process defects including single-sensor and multiple-sensor malfunction. These faults were induced in a Rapid Thermal Annealing (RTA) process, which is a common operation in microelectronics manufacturing. This process thermally treats semiconductor wafers after ion implantation. The

proposed FDD solution led to a fault reconstruction that was unbiased from both the normal process and the recorded fault magnitudes.

Evan et al. [54] compared all of PCA, Dynamic PCA (DPCA), and Canonical Variate Analysis (CVA)'s performances. They mainly focused on finding the best FDD technique with regards to robustness, sensitivity, and promptness. The researchers chose these three multivariate techniques because of their proven ability to detect multiple faults simultaneously. To establish the comparison grounds, the researchers simulated 21 various faults using the *Tennesse Eastman Process Simulator*. This study confirmed some general trends, such as the superiority of Q-statistics over T^2-statistics in picking up different faults (sensitivity). Added to that, this work demonstrated that both CVA's indices (T^2 *and* Q) recorded the smallest detection delay (promptness), while PCA and DPCA's indices gave the lowest false alarm rates (robustness). At last, the researchers concluded that both DPCA and PCA have similar FDD performances for the particular studied test, even though DPCA has the advantage of augmenting each observation vector with the previous recorded observations (dynamic update of samples).

Cho et al. [55] proved the superiority of Kernel PCA over normal PCA in monitoring nonlinear processes. To do so, the researchers defined two new statistics that reflect the contribution of both Hotteling's T^2, represented by C_{T^2} coefficient, and Squared Prediction Error Q, represented by C_Q coefficient. These new statistics were directly derived from the Kernel PCA formulation using the gradient kernel function. By doing so, these coefficients bypassed the need to do any approximation or data reconstruction procedure for fault identification that is generally required by the other type of nonlinear-PCA. In the end, these newly-established coefficients demonstrated satisfactory performances in detecting both ramp-type and bias-type faults in a simulated non-isothermal chemical process (CSTR).

2.2.5. Multi-Scale Principle Component Analysis and Mod-MSPCA

As described in the previous section, PCA is a great tool to reduce the dimensionality of the original data matrix by capturing the most important underlying variations and relationships between the variables. This characteristic makes it suitable for application involving multiple sensors. However, by itself, PCA is capable of feature extraction along only one localization, either in time or frequency; it lacks the capabilities to determine events with energy and power spectrums changing in both time and frequency. Another shortcoming to this technique is its limited capacity to reduce the error inherent in the original data matrix which deteriorates the reliability of the produced results. Unlike principal component analysis, wavelet transforms ensure the reduction of error by filtering the original data, and they have the ability to represent deterministic features with a small number of relatively large coefficients. Added to that, wavelets can easily isolate events in both time and frequency. However, they lack the capability to carry multivariable processing that is required by most industrial processes in order to distinguish sensor faults from process faults. That's why combining PCA and wavelets in one powerful method, known as Multiscale Principal Component Analysis (MSPCA) introduced by Bakshi [56], harnesses the advantages of each technique simultaneously. With that, MSPCA combines the extraction of deterministic features, done by wavelet transform, with the decorrelation and isolation of linear relationships which is carried by PCA. Figure 2.11 presents an overview of the technique where g and h are the low and high pass filters respectively.

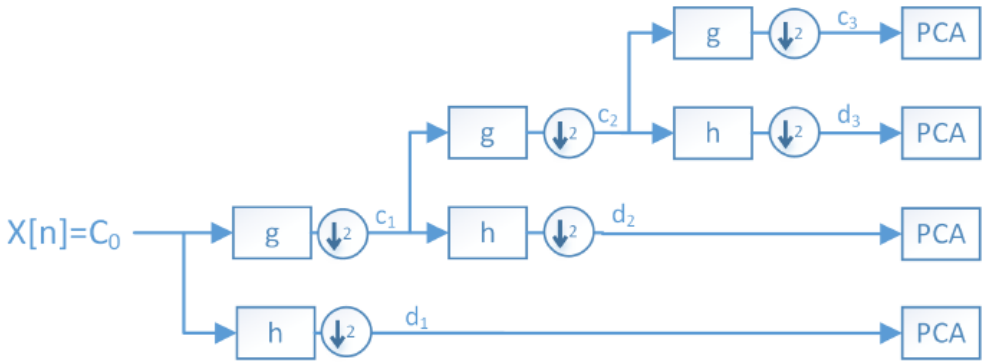


Figure 2.11 Schematic Diagram of a Three-Scale MSPCA Technique [9]

To explain the above Figure 2.11, each column in the measurement data matrix X undergoes the wavelet decomposition first. For each scale in the resulting decomposition, the wavelet coefficients serve as inputs to the PCA's covariance, SSCP, or correlation matrix. After that, both of PCA loading and scores matrices, T and P, are calculated for each wavelet coefficient. Finally, MSPCA builds the PCA in-control model (upper limits of $\mathcal{T}^2$, $\mathcal{Q}$, and φ) by selecting both wavelet coefficients and number of principal components through either Kaiser or Heuristic method. To sum up, process monitoring using MSPCA results in determining the independent PCs and their limits at each wavelet scale.

Although MSPCA provides many advantages over PCA and wavelet stand-alone techniques, it still lacks the ability to detect quickly and continuously the presence of shifts between operational states, e.g. abnormal to normal process state. In fact, Yoon and MacGregor [57] showed that MSPCA, due to the inherent scale decomposition, continues to flag false alarms even after the process has returned to its normal operations. Added to that, since the wavelet transform down-samples the signal, it can lead to numerical errors and statistical anomalies which explain the presence of these false alarms. To go around these limitations, the modified version of MSPCA, Mod-MSPCA [57],

adds a final step that combines up-scaling and Reconstruction Based Charts (RBC) which allows to diagnose the faults better. Furthermore, in this modified version, all produced wavelet coefficient are reconstructed using reconstruction filters before going as inputs to the PCA. Figure 2.12 describes this concept in details.

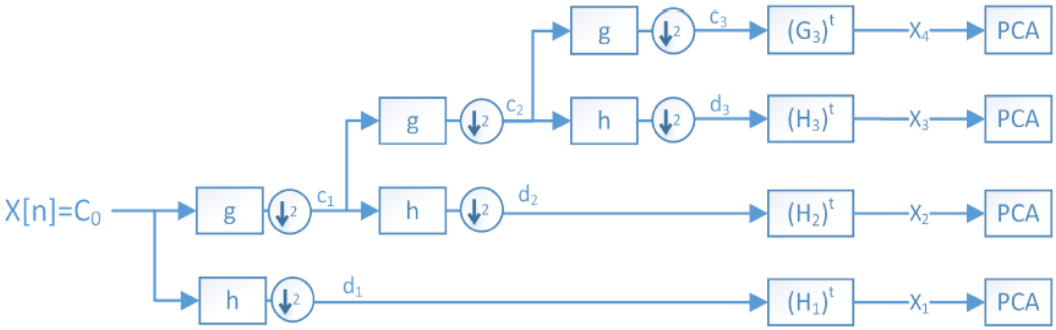


Figure 2.12 Schematic Diagram of a Three-Scale Mod-MSPCA Technique [9]

One common characteristics between MSPCA and Mod-MSPCA is present in the calculation of the final PCA statistical indices. In fact, these last are applied on each individual scale of the wavelet transform. Therefore, the previously defined $\mathcal{T}^2$, $\mathcal{Q}$, and φ in (2.18) ~ (2.21) can be rewritten as:

$$\mathcal{T}_j^2 = \sum_{i=1}^{A} \frac{\tau_i}{\lambda_{i,j}} \text{, where j = 1,2, ..., J+1} \tag{2.28}$$

$$\mathcal{Q}_j = \sum_{i=1}^{A} \tilde{x}_{i,j}^2 \text{, where j = 1,2, ..., J+1} \tag{2.29}$$

$$\varphi_j = \frac{\mathcal{Q}_j}{\delta_j^2} + \frac{\mathcal{T}_j^2}{\tau_j^2} \text{, where j = 1,2, ..., J+1} \tag{2.30}$$

Where J is the number of levels in the wavelet transform, A the number of principal components, and $\lambda_{i,j}$ eigenvalues of the i[th] variable's variance at the j[th] level. To recover the overall indices from the individual scales, the following three equations are employed:

$$\mathcal{T}^2 = \sum_{j=1}^{J} \mathcal{T}_j^2 \tag{2.31}$$

$$\mathcal{Q} = \sum_{j=1}^{J} \mathcal{Q}_j \tag{2.32}$$

$$\varphi = \frac{\mathcal{Q}}{\delta^2} + \frac{\mathcal{T}^2}{\tau^2}, \quad \text{where } \delta^2 = \sum_{j=1}^{J} \delta_j^2 \ \textit{and} \ \tau^2 = \sum_{j=1}^{J} \tau_j^2 \tag{2.33}$$

As mentioned before, unlike MSPCA, Mod-MSPCA takes these computed PCA statistics one step further by generating the reconstruction based charts. Dunia and Qin [58] showed that the combination of RBC with PCA results guarantees a higher rate of correct diagnosis. This is the case as faulty variables produce the highest contribution to the reconstructed results. To prove this, they modelled the faults using the equation $x = x^* + f$, where x^* reflects the normal operation sample vector, and f reflects the faulty vector defined as $f = \mathfrak{f}\,\xi_i$. The $m \times 1$ vector $\xi_i = [0 \dots 0\ 1\ 0 \dots 0]^T$, with the i[th] element equal to unity, is the fault direction vector. Consequently, the RBC technique aims at reconstructing x^* from every new sample vector x along all possible fault directions. The final minimum x^*direction, where f contribution is maximum, is used to identify the faulty variable. The RBC formulas combined with PCA's $\mathcal{T}^2$, $\mathcal{Q}$, and φ statistics are as follows:

$$RBC_{i,\mathcal{T}^2} = \frac{(x\,\hat{D}\,\xi_i)^2}{\hat{d}_{ii}} \text{ ,where } \hat{d}_{ii} = \xi_i{}^T \hat{D} \xi_i \; is\; the\; i^{th} diagonal\; elemenet\; of \hat{D} \qquad (2.34)$$

$$RBC_{i,\mathcal{T}^2} = \frac{(x\,\tilde{C}\,\xi_i)^2}{\tilde{c}_{ii}} \text{ , where } \tilde{c}_{ii} = \xi_i{}^T \tilde{C} \xi_i \; is\; the\; i^{th} diagonal\; elemenet\; of \tilde{C} \qquad (2.35)$$

$$RBC_{i,\varphi} = \frac{(x\,\Phi\,\xi_i)^2}{\Phi_{ii}} \text{ , where } \Phi_{ii} = \xi_i{}^T \Phi \xi_i \; is\; the\; i^{th} diagonal\; elemenet\; of \Phi \qquad (2.36)$$

In the above equations (2.34) ~ (2.36), $\hat{D}, \tilde{C}, and\ \Phi$ are the matrices defined in equations (2.18), (2.19), and (2.21) respectively. These equations ensure fault isolation through reconstruction-based contribution analysis. And by that, Mod-MSPCA is a powerful tool to detect, identify, and analyze different types of faults.

Yoon and MacGregor [57] highlighted the validity of Mod-MSPCA as a reliable FDD technique using a Monte Carlo simulation with a Continuous Stirred Tank Reactor (CSTR) system. A total of 500 sets of training and testing data were generated to cover different types of faults regarding the inlet temperature measurements. Mod-MSPCA performance was compared with PCA in order to detect and isolate sensor spikes, sensor drifts, sensor biases, and other complex phenomena like scaling effects. In the end, Mod-MSPCA was more effective than regular PCA methods, and it was specifically insightful for faults that are localized in frequency or appear in wavelet scales with small variances.

Lachouri [59] relied on MSPCA stand alone technique to monitor bearing faults. To put this method to the test, he used known recordings of both healthy and faulty bearing's vibrational signals. The results of this work demonstrated that most faults violated the 95% upper limit threshold for SPE statistics ($\mathcal{Q}$) which was used to signal their presence. Besides, Lachouri showed that SPE alone (without $\mathcal{T}^2$ statistics) can detect most of machinery faults that are related to bearings.

Wenying [60] combined MSPCA with a powerful classification technique using Support Vector Machines (SVM) to diagnose the health of induction motors. In this method, both $\mathcal{T}^2$ and Q results were fed directly as inputs to the SVM. To test the validity of the method, vibration data was collected from 4 accelerometers mounted on the motor in all 4 directions. The combination of MSPCA and SVM was very accurate in detecting and isolating 4 induced faults: abnormal stators, non-uniform air-gap, unbalanced rotor, and faulty rolling bearings.

2.2.6. Summary

This sections covered the fundamental mathematical tools that contribute to FDD diagnosis. All of Fourier Transform (FT), Wavelet Transform (WT), and Principal Component Analysis (PCA) theory and practical FDD applications were described in details. As these stand-alone techniques were limited, both MSPCA and Mod-MSPCA were introduced. By combining WT and PCA, the resulting techniques are reported to provide an increase in their feature extraction performances.

2.3. Neural Networks and Training Algorithms

This section starts by covering the basic concepts behind machine learning, notably the Multi-Layer Perceptron (MLP) architecture. It defines the steps that make up back propagation algorithm. This section concludes on three different training algorithms used in this research: Levenberg-Marquart (LM), Extended Kalman Filter (EKF), and Smooth Variable Structure Filter (SVSF).

2.3.1. Multi-layer Perceptron Neural Network

Like the human brain, neurons constitute the fundamental building bocks for Neural Networks (NN). In fact, clustering these neurons in one layer and connecting them to neurons of another layer makes up a Multi-Layer Perceptron (MLP) neural network. Figure 2.13 represent a common

nonlinear model of the neuron that contains m different inputs with their respective synaptic weights, a bias, and an activation function.

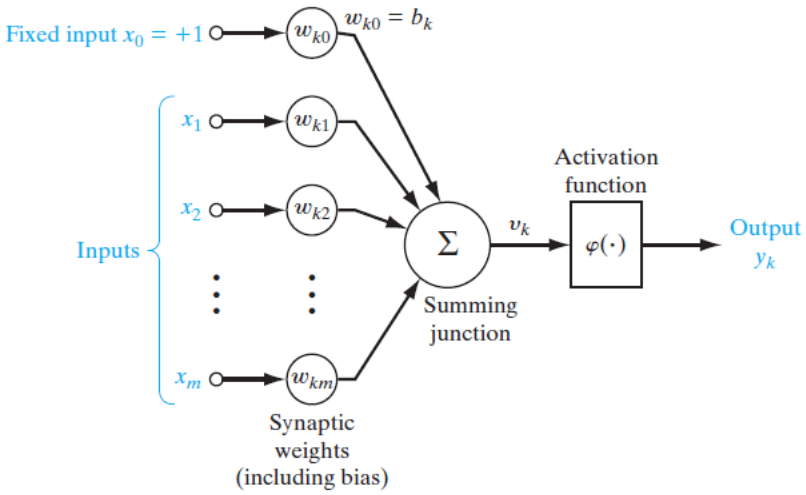


Figure 2.13 Nonlinear Model of a Neuron [61]

In the above model, both the inputs and the bias of the neuron get multiplied with their respective weights to be added together in the summing junction resulting in the signal $v_k = \sum_{j=0}^{m} w_{kj} x_j$. This signal acts as input for the activation function $\varphi(v_k)$ which produces the output of the neuron $y_k = \varphi(v_k)$. There are mainly two common activation functions generally used: sigmoid and tangent hyperbolic functions. Figure 2.14 demonstrates the differences between the two.

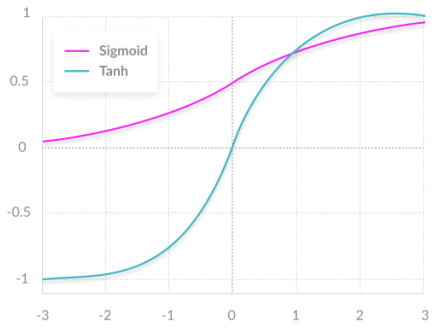


Figure 2.14 Two Common Neural Networks Activation Functions [62]

The above activation functions represent the nonlinear component in the neuron making it suitable for nonlinear applications. With that, depending on the activation function used, the output of the neuron y_k can be expressed as:

$$y_k = \frac{1}{1+\exp(-\alpha \sum_{j=0}^{m} w_{kj} x_j)}, \text{ when } \varphi(v_k) = \frac{1}{1+\exp(-\alpha v_k)}$$

Or

$$y_k = \tanh\left(\sum_{j=0}^{m} w_{kj} x_j\right), \text{ when } \varphi(v_k) = \tanh(v_k) \quad (2.37)$$

Given the neuron model, more complex structure can be built such as the MLP shown in Figure 2.15. The graph shows a fully connected NN with two hidden layers and one output layer that contain three nodes (neurons). The signal inside the MLP is indicated by the arrows' direction (from left to right).

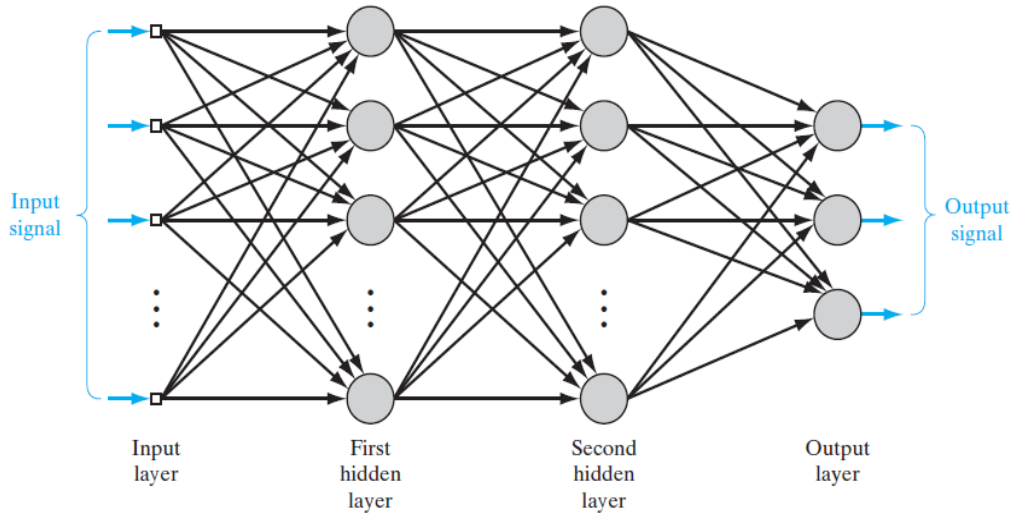


Figure 2.15 A Multilayer Perceptron With Two Hidden Layers [61]

To fully train such a structure for a particular application at hand, two types of signals have to be considered. The first type represent function signals that carry the forward propagation, from

neuron to neuron and from layer to layer, to transform all the input signals into a set of output values that reflect the network's estimate for that particular set of inputs. This estimate is then compared with the expected value of the output to generate an error signal which propagates backwards (layer by layer) from the output layer to the input layer. Figure 2.16 illustrates these function signals.

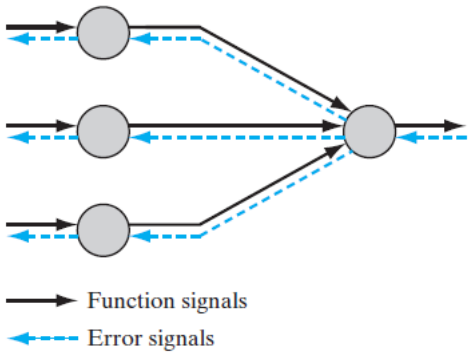


Figure 2.16 The Direction of Function and Error Signals in an MLP [61]

The propagation of both the function and the error signals represent the fundamental principle of supervised learning where each weight inside the network is readjusted based on the error signal from a training set produced at the output layer. This supervised learning presents two main types of learning methods: batch learning where the adjustments of synaptic weights is performed after all *N* training samples are fed to the network, and online learning where the weight adjustment is continuously done from one sample to another. What makes online learning the better candidate for the purpose of FDD is its inherent stochastic nature that prevent the network weights to be trapped in a local-minimum. Another added features is the lower requirement in both storage and computational capabilities that are usually limited in nature for most data acquisition (DAQ) systems.

To fully train a network using the online learning method, there are 5 main steps that have to be performed in sequence. These steps constitute the back-propagation learning from the training sample $\{x(n), d(n)\}_{n=1}^{N}$ where $x(n)$, $d(n)$, and N are the input, the expected output , and the number of training samples respectively. These steps are summarized as follow:

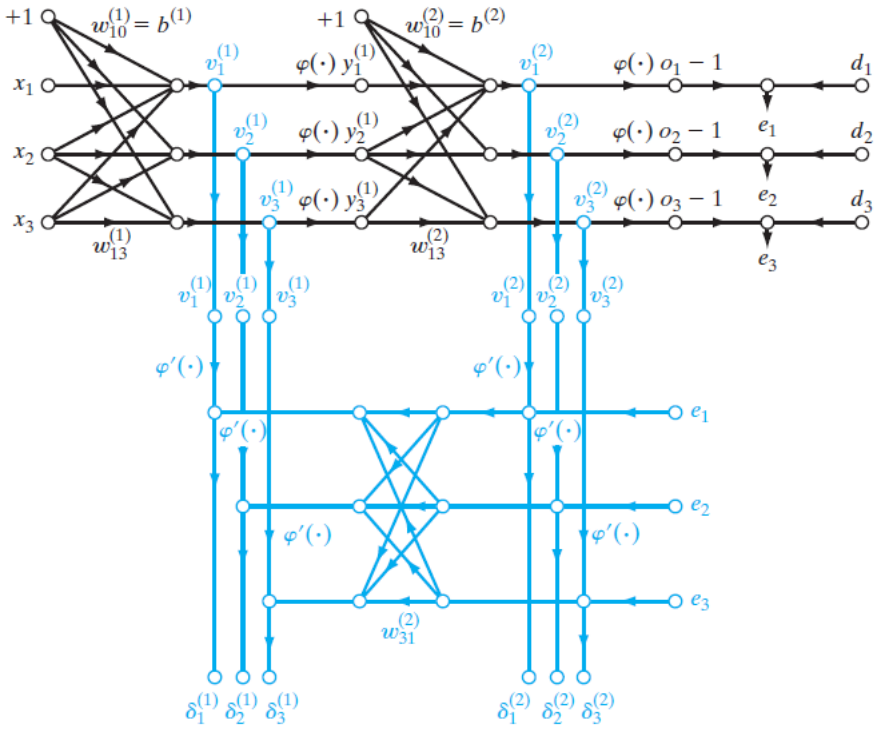


Figure 2.17 Signal Flow of Back-propagation Learning [61]

I. Initialization Step: initialize all the weights in the network using a uniform distribution with a zero-mean and a variance. This ensures that every neuron's local field v_k is lying in the transitional part of either of sigmoid or tangent hyperbolic activation functions.

II. Training Samples Preparation Step: order the training set either randomly or according to a certain rule. This step presents the network with the predetermined sequence of samples that is used for training purposes.

III. Forward Computation Step: feed the MLP's first layer (Input layer) with the input $x(n)$ extracted from the training set $\{x(n), d(n)\}_{n=1}^{N}$. After that, using the equation (2.38), compute the local field $v_j^{(l)}(n)$ produced by every neuron j in the layer l at the time step n. Once the local field in obtained, compute the output of every neuron using the appropriate activation function, see equation (2.39). This step is repeated for all layers present in the network where $l \in [1, L]$. The final layer produces results in the output of the MLP $o_j(n)$ for a particular input signal $x(n)$. This computed output is then compared with the desired response vector $d_j(n)$ to produce an error signal described by equation (2.40).

$$v_j^{(l)}(n) = \sum_i w_{ji}^{(l)}(n) y_i^{(l-1)}(n) \tag{2.38}$$

Where

$$y_j^{(l)} = \varphi_j\left(v_j(n)\right), y_j^{(0)} = x_j(n), and\ y_j^{(L)} = o_j(n), \tag{2.39}$$

And :

$$e_j(n) = d_j(n) - o_j(n) \tag{2.40}$$

IV. Backward Computation Step: determine the local gradients and update the weights accordingly. Each local gradient, $\delta_j^{(l)}(n)$, is computed using equation (2.41); at every layer l and for every neuron j the backward-propagating error signal is multiplied with the

derivation of the activation function of the local field $\varphi_j'\left(v_j^{(L)}(n)\right)$. Once the local gradients are obtained, the synaptic weights at each layer of the MLP are updated in order to adjust for the intensity of the error signal, see equation (2.42). This adjustment is tweaked further using two additional terms of momentum constant α in the term: $\alpha\ \left[\Delta w_{ji}^{(l)}(n-1)\right]$, and learning rate η in: $\eta\ \delta_j^{(l)}(n)y_i^{(l-1)}(n)$.

$$\delta_j^{(l)}(n) = \begin{cases} e_j^{(L)}(n)\varphi_j'\left(v_j^{(L)}(n)\right), & j = L\ (output\ layer) \\ \varphi_j'\left(v_j^{(l)}(n)\right)\sum_k \delta_k^{(l+1)}(n)w_{kj}^{(l+1)}(n), & j = l\ (hidden\ layer) \end{cases} \quad (2.41)$$

$$w_{ji}^{(l)}(n+1) = w_{ji}^{(l)}(n) + \alpha\ \left[\Delta w_{ji}^{(l)}(n-1)\right] + \eta\ \delta_j^{(l)}(n)y_i^{(l-1)}(n) \quad (2.42)$$

V. Iteration Step: iterate both forward and backward computation steps until one of the stopping conditions is met, e.g. minimum local gradient or minimum RMS error.

Through the above example of weight adjustment, neural networks are able to learn and store information about different processes via their associative memory. This diagnostic ability allows such tools to classify non-linearly separable data. Thus, it is suitable for FDD applications. Samanta [63] compared the performance of both support vector machines with MLP architectures in detecting different gear faults. To do so, the generated gearbox vibration signals were used. From these recordings, different features were extracted using Genetic Algorithm (GA). These features included, mean, variance, root mean square (*rms*), and kurtosis and were applied to distinguish between healthy and defective gears. Furthermore, Samanta used a three layer MLP network to investigate its general classification capabilities under different sensor locations, signal pre-processing features, and sampling rates. This work demonstrated that the classification accuracy of both MLP and SVM were spot-on 100% and comparable with each other especially when an

additional step of pre-processing is applied. Nevertheless, without genetic algorithm feature extraction for both classifiers obtained an accuracy above 97%.

Jack and Nandi [64] carried a similar research by comparing SVM and MLP's ability to detect defective roller bearings. Four different faults were introduced to the machine while it was running at constant speed: inner race fault, outer race fault, rolling element fault, and cage fault. A total of 156 different features (90 statistical and 56 spectral) were extracted from a biaxial accelerometer mounted on the machine. The results from the training demonstrated that having a high number of features correlates with having a network with poorer generalization performance. This is mainly due to the overlap that exists between certain features that confuses both MLP and SVM classifiers. To curve this problem, the researchers filtered the number of features using genetic algorithm and were able to increase the accuracy of both classifiers up to 99%.

Kajiro et al. [65] implemented a two-stage multilayer neural network, see Figure 2.18, to detect typical faults in chemical processes, such as fouling of the heat exchanger surface, physical deterioration of the catalyst, partial plugging of the pipeline, and decrease in the heater performance. In this two stage network, the first stage is used to classify between different types of faults, while the second stage is used to estimate the level of deterioration (from 1 to 5) of the classified faults. To train the network, the researcher relied on the aforementioned 5-step method to carry back propagation algorithm (learning rate constant η = 0.1, and momentum constant α =0.9, in equation (2.42)). All of the outlet concentration, the heater outlet's temperature, and the controller's output signal were used as inputs to train both the first and the second stage 3-layer MLP architectures. By doing so, the researchers were able to reduce the amount of calculation needed when compared to one big multilayer neural network. As a matter of fact, if the number of deterioration levels or the number of faults were to change, the two stage approach does not require training the whole network from the beginning; the acquired old knowledge about the system would

be still viable for classification. By that, this research demonstrated that multistage detection process using NNs is a robust and accurate classification technique to do fault diagnostics in noisy environments, such as chemical reactors.

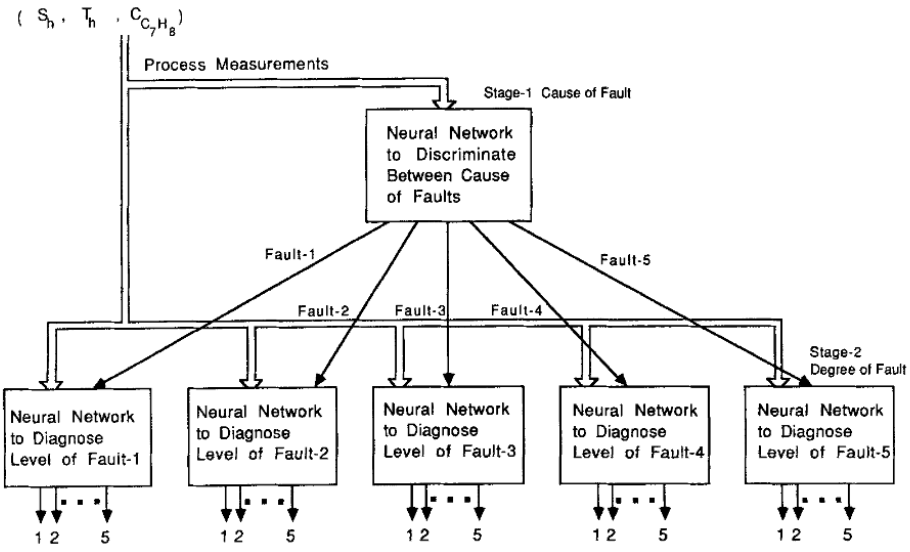


Figure 2.18 Two-stage Artificial Neural Network for Fault Diagnosis of Chemical Processes [65]

2.3.2. Neural Networks Training Algorithms

To be able to use neural networks as a viable classifier tool, the weights have to be trained to meet the requirements of the application at hand. This section covers three different training algorithms of the MLP networks, and they are: Levenberg-Marquart (LM), Extended Kalman Filter (EKF), and Smooth Variable Structure Filter (SVSF). Within each subsection, the main driving equations and the training steps are outlined in details.

Levenberg-Marquart (LM)

The back-propagation algorithm (gradient descent) remains one of the most important training methods for multi-layer perceptron architectures in neural networks. It employs the steepest descent approach where the step size for updating the weights, see equation (2.42), is always constant regardless of the gradient (reflect by $\Delta w_{ji}^{(l)}(n-1)$). This makes the algorithm inefficient and slow to converge. In fact, to obtain a stable convergence towards the minima, the steepest descent employs small increments to update the elements of the weight vector. This allows to reduce any unwanted oscillations around the minima that usually occur due to the presence of steep (large) gradients. And these small updates directly translate in a slow convergence rate of the algorithm which constitute its main limitation. To make this method faster, Levenberg and Marquardt [66] [67] proposed a solution that blends the gradient descent with a faster algorithm: Gauss-Newton method. This method employs the second order derivative of the sum square error (SSE) function, $E(x,w)$, to detect the proper step size of every error direction and to converge faster. With that, this combined implementation inherits the stability of gradient descent along with the speed of Gauss-Newton and can even guarantee convergence at the first iteration when the error function $E(x,w)$ has a quadratic surface [68].

To derive the Levenberg-Marquardt algorithm, the global error function used to update the weights needs to be defined. As mentioned earlier, Sum Square Error (SSE) function is a common type of function used in the training process and it is evaluated as follow:

$$E(x,w) = \frac{1}{2}\sum_{n=1}^{N}\sum_{m=1}^{M} e_m(n)^2 \tag{2.43}$$

Where $x, w, N, and\ M$ are respectively the input vector, the weight vector, the number of examples in the training set, and the number of output nodes . $e_m(n)$ is defined in equation (2.40), and it represent the training error at output m when the training sample with index n from the set

$\{x(n), d(n)\}_{n=1}^{N}$ is provided. From this error function, the gradient g, which uses the first-order derivative of the total error function with respect to the weight vector, is defined as such:

$$g = \frac{\partial E(x,w)}{\partial w} = \left[\frac{\partial E}{\partial w_1} \;\; \frac{\partial E}{\partial w_2} \;\; \dots \frac{\partial E}{\partial w_P}\right]^T \tag{2.44}$$

In the above equation P represent the total number of weights in the Neural Network. With this new definition of the gradient g, the gradient descent update equation (2.42) can be simplified as such:

$$w(n+1) \approx w(n) - \alpha g(n) \qquad (\alpha \text{ learning constant}) \tag{2.45}$$

Following on this, Newton's method [68] has shown that the gradient g in (2.44) and the weight difference Δw in (2.42) are related after expanding the Taylor series approximation of the equation (2.44), deriving the results with respect to the weight vector (∂w), and solving it to obtain the minima of the total error function $E(x,w)$. This relationship is defined by the following equation (2.46) where the matrix H is called the Hessian matrix and it is defined in equation (2.47):

$$-g = H\Delta w \;\leftrightarrow\; \Delta w = -H^{-1}g \tag{2.46}$$

Where

$$H = \begin{bmatrix} \frac{\partial^2 E}{\partial w_1^2} & \frac{\partial^2 E}{\partial w_1 \partial w_2} & \dots & \frac{\partial^2 E}{\partial w_1 \partial w_P} \\ \frac{\partial^2 E}{\partial w_2 \partial w_P} & \frac{\partial^2 E}{\partial w_2^2} & \dots & \frac{\partial^2 E}{\partial w_2 \partial w_P} \\ \dots & \dots & \dots & \dots \\ \frac{\partial^2 E}{\partial w_P \partial w_1} & \frac{\partial^2 E}{\partial w_P \partial w_2} & \dots & \frac{\partial^2 E}{\partial w_P^2} \end{bmatrix} \tag{2.47}$$

Levenberg-Marquardt Algorithm updates the neural network's weights using the inverse of the above Hessian matrix. However, deriving both the Hessian and its inverse is very computationally expensive due to the presence of partial second derivatives. To curve this problem, many approximation to the Hessian matrix have been provided, such as the Gaussian method. However, unlike the other methods, Levenberg-Marquardt approach [68] ensures that both the Hessian and its inverse are always definite and easy to compute. And by that, the Hessian matrix can be approximated using the outer product of a new matrix: The Jacobian matrix J, see equation (2.48) and (2.49). Therefore, the LM technique, which is well designed for nonlinear least-square estimation problems, approximates computing the second-order derivatives of the total error function in the Hessian matrix by relying solely on the first-order derivatives of the individual error signal defined by the Jacobian matrix.

$$H \approx J^T J + \mu I, \qquad I: Identity\ matrix \tag{2.48}$$

Where:

$$J = \begin{bmatrix} \frac{\partial e_1(1)}{\partial w_1} & \frac{\partial e_1(1)}{\partial w_2} & \dots & \frac{\partial e_1(1)}{\partial w_P} \\ \frac{\partial e_2(1)}{\partial w_1} & \frac{\partial e_2(1)}{\partial w_2} & \dots & \frac{\partial e_2(1)}{\partial w_P} \\ \dots & \dots & \dots & \dots \\ \frac{\partial e_M(1)}{\partial w_1} & \frac{\partial e_M(1)}{\partial w_2} & \dots & \frac{\partial e_M(1)}{\partial w_P} \\ \dots & \dots & \dots & \dots \\ \frac{\partial e_1(N)}{\partial w_1} & \frac{\partial e_1(N)}{\partial w_2} & \dots & \frac{\partial e_1(N)}{\partial w_P} \\ \frac{\partial e_2(N)}{\partial w_1} & \frac{\partial e_2(N)}{\partial w_2} & \dots & \frac{\partial e_2(N)}{\partial w_P} \\ \dots & \dots & \dots & \dots \\ \frac{\partial e_M(N)}{\partial w_1} & \frac{\partial e_M(N)}{\partial w_2} & \dots & \frac{\partial e_M(N)}{\partial w_P} \end{bmatrix} \tag{2.49}$$

In the above equation (2.48), the additional term μI in the Hessian matrix approximation guarantees that the matrix is always positive definite. This is a necessary mathematical condition

when computing the inverse $H(n)^{-1}$ in the update equation (2.50) for the MLP weights using Levenberg-Marquardt algorithm:

$$w(n+1) = w\,(n) - \,H(n)^{-1}J(n)e(n) \tag{2.50}$$

To sum up, the procedure in training an MLP using LM algorithm is similar to the back propagation aforementioned 5 steps. Except for step 4, the Backward Computation Step, which encompasses and additional Jacobian and inverse-Hessian matrix computation, all the other steps are identical with back propagation algorithm. The Levenberg-Marquardt algorithm constitute a fast and stable combination between steepest descent and Gauss-Newton algorithms.

Extended Kalman Filter (EKF)

The introduction of the Kalman filter in 1960 provided an efficient computational recursive solution of the least-squares method. However, Kalman Filter (KF) is only relevant to linear state and parameter estimation problems subject to white noise assumptions. For nonlinear problems, the Extended Kalman Filter (EKF) has been proposed. EKF tackles state estimation for nonlinear processes by linearizing the system model using a prediction of the current mean and covariance matrices.

EKF assumes that systems are modeled using the state-space representation as follows:

$$x_k = f(x_{k-1}, u_k) + \,w_k \tag{2.51}$$

$$z_k = h(x_k) + \,v_k \tag{2.52}$$

Where w_k (process noise) and v_k (measurement noise) are multivariate Gaussian noises with zero mean and covariance matrices $Q_k \; and \; R_k$ respectively. u_k represents the input control vector of the system. Both f and h are nonlinear functions that are used in getting both the predicted state

and the predicted measurement respectively. These functions are approximated with their partial derivatives by computing the Jacobian matrices, F_k& H_k, defined as such:

$$F_k = \left.\frac{\partial f}{\partial x}\right|_{\hat{x}_{k-1|k-1}, u_k} \tag{2.53}$$

$$H_k = \left.\frac{\partial h}{\partial x}\right|_{\hat{x}_{k|k-1}} \tag{2.54}$$

The Discrete EKF governing equations can be clustered in to prediction and update equations. Prediction equations are as follows, see table of nomenclature:

$$\hat{x}_{k|k-1} = f(\hat{x}_{k-1|k-1}, u_k) \tag{2.55}$$

$$P_{k|k-1} = F_k P_{k-1|k-1} F_k^T + Q_k \tag{2.56}$$

Following the prediction step are the update equations, and they should be computed in the following sequence:

$$\tilde{y}_k = z_k - h(\hat{x}_{k|k-1}) \tag{2.57}$$

$$S_k = H_k P_{k|k-1} H_k^T + R_k \tag{2.58}$$

$$K_k = P_{k|k-1} H_k^T S_k^{-1} \tag{2.59}$$

$$\hat{x}_{k|k} = \hat{x}_{k|k-1} + K_k \tilde{y}_k \tag{2.60}$$

$$P_{k|k} = (I - K_k H_k) P_{k|k-1} \tag{2.61}$$

The EKF can be used to estimate the weights of the neural network which represent a nonlinear dynamic system. To carry this training, it is necessary to rearrange all inputs, outputs, and network weights in the state space format. For the following training sample set $\{x(n), d(n)\}_{n=1}^{N}$ where $x(n)$ and $d(n)$ are the input and output to the MLP network respectively, the MLP neural network can be represented by the following two equation:

$$w_{n+1} = w_n + \omega_n \tag{2.62}$$

$$d(n) = g\big(w_n, x(n)\big) + v_n \tag{2.63}$$

Where w_n denotes the parameters of the MLP, e.g. weights and biases between the nodes, g represent the measurement function which is defined by equation (2.37), and both v_n and ω_n denote white Gaussian noises with zero mean and with covariance matrices $R_n \ and \ Q_n$ respectively.

The EKF-based training is summarized by the following 8 steps:

I. Initialize all the weight estimates $\widehat{w}_{n|n}$ of the MLP with random values $\in [-1,1]$.

II. Compute the predicted state's Jacobian matrix approximation F_{n+1} using:

$$F_{n+1} = \left.\frac{\partial f}{\partial w}\right|_{\widehat{w}_{n|n}, u_{n+1}} \tag{2.64}$$

III. Compute the predicted a posteriori weight ($\widehat{w}_{n+1|n}$) and covariance matrix ($P_{n+1|n}$) estimates:

$$\widehat{w}_{n+1|n} = \widehat{w}_{n|n} \tag{2.65}$$

$$P_{n+1|n} = F_{n+1} P_{n|n} F_{n+1}^T + Q_{n+1} \tag{2.66}$$

IV. Compute the measurement function's Jacobian matrix H_{n+1} using the obtained a posteriori weight ($\widehat{w}_{n+1|n}$):

$$H_{n+1} = \left.\frac{\partial h}{\partial w}\right|_{\widehat{w}_{n+1|n}} \tag{2.67}$$

V. Obtain the measurement residual error ($\tilde{e}_{n+1|n}$) and the residual covariance matrix (S_{n+1}) using:

$$\tilde{e}_{n+1|n} = d(n+1) - g\left(\widehat{w}_{n+1|n}, x(n+1)\right) \tag{2.68}$$

$$S_{n+1} = H_{n+1} P_{n+1|n} H_n^T + R_n \tag{2.69}$$

VI. Obtain the EKF gain using:

$$K_{n+1} = P_{n+1|n} H_{n+1}^T S_{n+1}^{-1} \tag{2.70}$$

VII. Update both the state estimate and the covariance estimate using:

$$\widehat{w}_{n+1|n+1} = \widehat{w}_{n+1|n} + K_{n+1} \tilde{e}_{n+1|n} \tag{2.71}$$

$$P_{n+1|n+1} = (I - K_{n+1}H_{n+1})P_{n+1|n} \tag{2.72}$$

VIII. Compute the mean square error (MSE) using equation (2.73) . Repeat the sequence of step starting from step (II) until the MSE error is below a certain training threshold.

$$\tilde{e}_{n+1|n+1} = d(n+1) - g\left(\hat{w}_{n+1|n+1}, x(n)\right) \tag{2.73}$$

By considering the weights as dynamic, EKF can be used to estimate the weights of a neural network. This being said, linearization in both F_n& H_n Jacobian matrices introduces inherent errors in the computed weight estimates. This limits the performance of the EKF. That is why, a more robust estimator may be considered such as the Smooth Variable Structure Filter (SVSF).

Smooth Variable Structure Filter (SVSF)

The EKF's robustness and stability can be improved by combining it with the Smooth Variable Structure Filter (SVSF). This nonlinear filter was proposed by Habibi [69], in 2007, and it requires the system to be differentiable and hence the word "smooth". SVSF is based on Sliding Mode Control (SMC). Similarly, it defines a hyperplane of state trajectory, known as sliding surface, as shown in Figure 2.19. In its basic form, this filter forces the estimated states to the hyperplane by using a switching technique that causes the estimated states to cross this hyperplane continuously and remain confined to its neighborhood referred to as the existence subspace. This method guarantees convergence given bounded uncertainties.

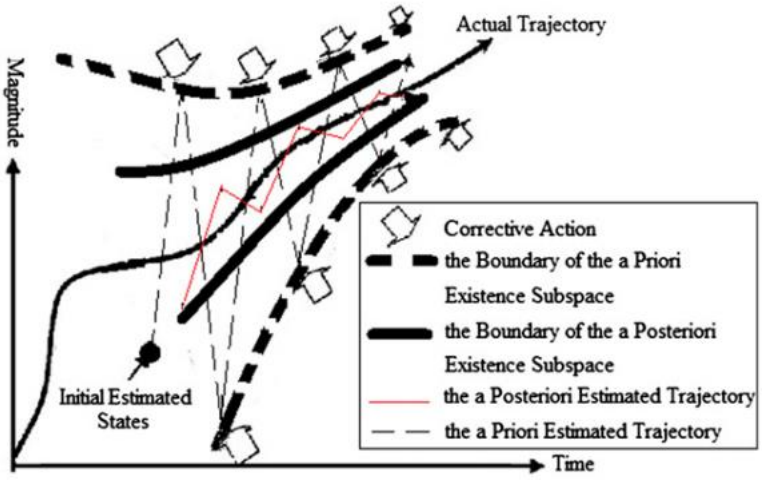


Figure 2.19 Concept of SVSF State Estimation [70]

SVSF presents two main performance indicators: the a posteriori output error and the chattering. This chattering can be sorted into two main categories: a priori and a posteriori chattering. At its core, SVSF's structure contains both prediction and update stages, shown in Figure 2.20.

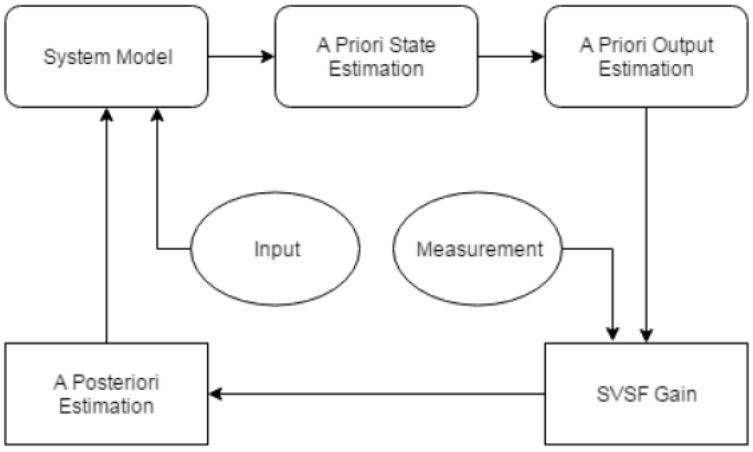


Figure 2.20 SVSF Structure [69]

The SVSF assumes that systems are modeled using the state space representation as follows:

$$x_{k+1} = f(x_k, u_k) + w_k \tag{2.74}$$

and

$$z_{k+1} = Cx_{k+1} + v_k \tag{2.75}$$

Where x_k and x_{k+1} are the system states at time step k and $k+1$ respectively. z_{k+1} is the measurement vector at time step $k+1$. u_k, w_k, and v_k are the system input, system noise, and measurement noise respectively. The matrix C represent the measurement matrix and it is considered linear, positive, and pseudo diagonal when the system has a linear measurement equation. In case the measurement equation becomes nonlinear, e.g. $z_{k+1} = g(x_{k+1}) + v_k$, the measurement matrix is linearized using a Jacobian matrix which is derived as follows:

$$C = \begin{bmatrix} \frac{\partial g_1}{\partial x_1} & \frac{\partial g_1}{\partial x_2} & \cdots & \frac{\partial g_1}{\partial x_n} \\ \frac{\partial g_2}{\partial x_1} & \frac{\partial g_2}{\partial x_2} & \cdots & \frac{\partial g_2}{\partial x_n} \\ \vdots & \vdots & \vdots & \vdots \\ \frac{\partial g_m}{\partial x_1} & \frac{\partial g_m}{\partial x_2} & \cdots & \frac{\partial g_m}{\partial x_n} \end{bmatrix} \tag{2.76}$$

The SVSF filter is an iterative algorithm that starts by predicting the a priori estimate of both the state and the output measurement of the system:

$$\hat{x}_{K+1|k} = f(\hat{x}_{k|k}, u_k) \tag{2.77}$$

$$\hat{z}_{K+1|k} = C\hat{x}_{K+1|k} \tag{2.78}$$

From the predicted a priori estimates, the SVSF filter updates the a posteriori estimates of both the states and the measurements, as follows:

$$\hat{x}_{K+1|k+1} = \hat{x}_{K+1|k} + K_{k+1}^{SVSF} \tag{2.79}$$

$$\hat{z}_{K+1|k+1} = C\hat{x}_{K+1|k+1} \tag{2.80}$$

The SVSF gain, K_{k+1}^{SVSF}, is obtained as:

$$K_{k+1}^{SVSF} = C^{+}\left(\left|e_{z,k+1|k}\right|_{abs} + \gamma\left|e_{z,k|k}\right|_{abs}\right) \circ sat\left(\frac{e_{z,k+1|k}}{\psi}\right) \tag{2.81}$$

Where

$$sat\left(\frac{e_{z,k+1|k}}{\psi}\right) = \begin{cases} sgn\left(e_{z,k+1|k}\right), & if\ \left|e_{z,k+1|k}\right| > \psi \\ \dfrac{e_{z,k+1|k}}{\psi}, & if\ \left|e_{z,k+1|k}\right| \leq \psi \end{cases} \tag{2.82}$$

and

$$e_{z,k+1|k} = z_{k+1} - \hat{z}_{k+1|k} \tag{2.83}$$

$$e_{z,k|k} = z_k - \hat{z}_{k|k} \tag{2.84}$$

$$C^{+} = C^{T}(CC^{T} + \rho^{2}I)^{-1} \text{ (pseudo inverse of } C) \tag{2.85}$$

In the above equations, ψ, ρ, and γ are respectively the smoothing boundary layer's width, singular values' damping parameter, and a positive constant less than unity. As described by equation (2.81), inside the smoothing boundary layer, the corrective action (SVSF gain) depends on the ratio of the amplitude of the output's a priori estimation error and the smoothing boundary layer's width, ψ. This width plays a key role in the overall performance of the filter and should be selected carefully. In fact, while larger widths directly translate into slower convergence rates, smaller widths cause unwanted chattering that degrade the filter's performance.

The purpose of training of the network is to determine its weights. Ahmed [71] demonstrated the ability of SVSF to train the MLP's weights. In order to use SVSF with a training set $\{x(n), d(n)\}_{n=1}^{N}$, the MLP neural network is represented by the following two equation:

$$w_{n+1} = w_n + \omega_n \tag{2.86}$$

$$d(n) = C_n\big(w_n, x(n)\big) + v_n \tag{2.87}$$

Where w_n denotes the parameters of the MLP, e.g. weights and biases between the nodes, C_n represent the measurement function, and both v_n and ω_n are white Gaussian noises with zero mean. Training the MLP using SVSF involves 9 different steps, and they are as follows:

I. Initialize all the weight estimates $\widehat{w}_{n|n}$ of the MLP with random values $\in [-1,1]$.

II. Deduce the predicted a posteriori weight estimates $\widehat{w}_{n+1|n}$ using:

$$\widehat{w}_{n+1|n} = \widehat{w}_{n|n} \tag{2.88}$$

III. Compute the Jacobian matrix $C_{n|linearized}$ as follows:

$$C_{n|linearized} = \begin{bmatrix} \frac{\partial d(1)}{\partial w_1} & \frac{\partial d(1)}{\partial w_2} & \cdots & \frac{\partial d(1)}{\partial w_P} \\ \frac{\partial d(2)}{\partial w_1} & \frac{\partial d(2)}{\partial w_2} & \cdots & \frac{\partial d(2)}{\partial w_P} \\ \vdots & \vdots & \vdots & \vdots \\ \frac{\partial d(N)}{\partial w_1} & \frac{\partial d(N)}{\partial w_2} & \cdots & \frac{\partial d(N)}{\partial w_P} \end{bmatrix} \tag{2.89}$$

Where P is the total number of connected node weights in the network.

IV. Generate the MLP network estimated output $\hat{d}(n+1|n)$ using:

$$\hat{d}(n+1|n) = C_{n|linearized}\widehat{w}_{n+1|n} \tag{2.90}$$

V. Calculate the measurement error $e_{n+1|n}$ using:

$$e_{n+1|n} = d(n) - \hat{d}(n+1|n) \tag{2.91}$$

VI. Compute the SVSF Gain using the following equation:

$$K_{n+1}^{SVSF} = C_{n|linearized}^{+}\left(\left|e_{n+1|n}\right|_{abs} + \gamma\left|e_{n|n}\right|_{abs}\right) \circ sat(\frac{e_{n+1|n}}{\psi}) \tag{2.92}$$

VII. Get the new State Estimates $\widehat{w}_{n+1|n+1}$ using:

$$\widehat{w}_{n+1|n+1} = \widehat{w}_{n+1|n} + K_{n+1}^{SVSF} \tag{2.93}$$

VIII. Obtain the updated state estimates and error.

$$\hat{d}(n+1|n+1) = C_{n|linearized}\widehat{w}_{n+1|n+1} \tag{2.94}$$

$$e_{n+1|n+1} = d(n+1) - \hat{d}(n+1|n+1) \tag{2.95}$$

IX. Calculate the Mean Square Error (MSE) and repeat the sequence of step starting from step (II) until the MSE error is below a certain training threshold.

2.4. Summary

A review of the different fault detection and diagnosis techniques was presented in this chapter. While some of these methods were suited for time-frequency analysis, others were useful for multi-sensory set-ups. To improve the accuracy of feature extraction, hybrid techniques, such as MSPCA and Mod-MSPCA, were discussed and their respective practical applications were reviewed. This chapter considered neural networks as well as the different algorithms used for their training. The combination of feature-extraction techniques along with feature-classification methods constitutes a valid FDD scheme suited for internal combustion engines. There are several ways to realize such combinations. Some of them are discussed in the following Chapter 3.

Chapter 3. FDD Strategies for ICE

This chapter builds on the concepts considered in the previous chapter under literature review. Three additional concepts that combine the basic tools used in FDD are discussed and further presented as follows:

- Method 1 (M1): the Average Crank-Angle Domain transform (M1-Average-CAD).
- Method 2 (M2): the Crank-Angle Domain transform of Industrial Extended Multi-Scale Principal Component Analysis (M2-CAD-IEMSPCA).
- Method 3 (M3): the Crank-Angle Speed Analysis (M3-CASA).

While the first two, notably M1-Average-CAD and M2-CAD-IEMSPCA, were repurposed for the ICE application and rely heavily on neural networks for classification, M3-CASA was introduced in the work and bypasses the classification step which makes it the fastest among the three. Nevertheless, all the implemented techniques apply the crank-angle transformation for maximum reliability.

3.1. FDD Strategies for ICE

The internal combustion engine technologies have been around for more than a century. Therefore, the literature is rich with different associated FDD techniques. What makes these methods differ from one another is the choice of the sensor used for perception and the mathematical tools used for analysis. With that, this section covers briefly other potential solutions that carry engine diagnostics.

Chang et al. [72] introduced a new method for misfire and knock detection using wavelet transform of high frequency engine block vibration signals. They defined a new Combustion Noise Intensity

factor CNI_{CWT} that is derived from the continuous-time wavelet transform's scalogram; this factor averages the absolute value of the CWT for a specific scale factor s and a crank-angle interval (θ_1, θ_2), see equation (3.1). The scale of the CWT was fixed to 8 and the mother wavelet used was the Meyer wavelet. To test the validity of this new factor, the researcher used different engine conditions (rpm, load, and faulty cylinders) and detected both misfire and knock using high frequency vibration signals of one-axial accelerometer which was attached to the engine block. Although CNI_{CWT} threshold was kept fixed across all the engine conditions, it detected both aforementioned faults with an overall accuracy of 96.7%. In conclusion, this method is more efficient and accurate than FFT power spectrum density techniques. It can detect misfire and knock without the need of an additional tuning stage.

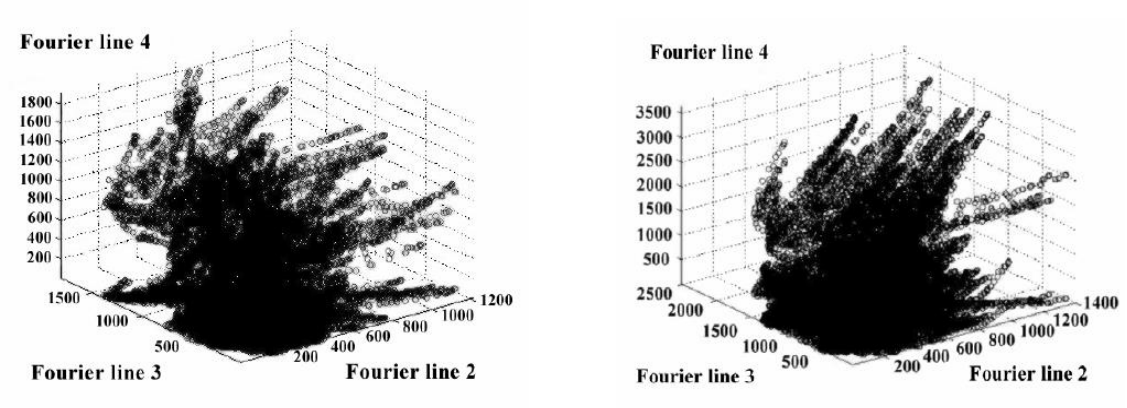
$$CNI_{CWT}(s, \theta_1, \theta_2) \equiv \frac{\int_{\theta_1}^{\theta_2} | W_{accelerometer}(t(\theta), s)| d\theta}{\theta_2 - \theta_1} \tag{3.1}$$

Daniels et al. [73] detected both heavy (audible) knock and light (inaudible) knock using in-cylinder ionization signal under different engine speeds and loads. To have a comparative study, this sensor's detection capabilities were measured against a knock sensor, a mounted accelerometer, and an in-cylinder pressure sensor. To obtain the test results, the researchers used a 2L, 4-cylinder engine and sampled the data using the dSPACE PX-10. At 5000 rpm and under WOT (Wide Open Throttle) conditions, all of knock, accelerometer, and pressure sensors failed to detect the induced knock as the recorded noise levels were high. Under the same conditions, ionization sensors provided a better detection of inaudible knock. What is more, the ionization sensor successfully detected partial misfire (partial-burn) conditions especially at cold start conditions of the engine. This fault is common during the expansion stroke and can result in increase of the HC (Hydro-carbon) emissions. Therefore, using a FDD technique that applies ionization sensor increased the

engine efficiency and limits the harm to engine components. In conclusion, the ionization signal provides a lot of valuable insights regarding the combustion process inside the cylinder and allows for an increased speed-range for fault detection when compared with conventional methods.

Bogus and Merkisz [74] used an analogous technique to Short Time Fourier Transform (STFT). The technique applies a small sliding-window to extract different statistical parameters used in a multidimensional parameter space in order to highlight the time evolution of the system (engine), see equation (3.2). The parameters can be either FFT lines or statistical-in-nature like median, mean, and higher order moments. Some of these parameters were nonlinear as well, like Lyapunov exponents and correlation factors. Once the multidimensional space is created, the fault condition is detected using the c-mean clustering algorithm that optimizes the objective function which depends on the Euclidian distance between a particular data point and the center of the cluster. To test the validity of this technique, the researchers simulated a misfire condition on a 12V, one-cylinder engine and logged the data using a tri-axial accelerometer. The test was carried for three main engine speeds: 650 rev/min (at idle), 1100 rev/min, and 1500 rev/min. After that, 7 misfire-detection parameters were selected among 5 FFT-lines, 5 moments, a mean, and a median. Figure 3.1 shows an example of a 7 dimensional parameter space that was projected onto a 3D space (Fourier lines 2, 3, and 4). The figure highlights the evolution of the computed parameters between a misfire condition (a) and a healthy condition (b). The final step assigns the different parameter clusters to different engine states in order to detect misfire condition of diesel engines.

$$X(f,n) = \sum_{k=-\infty}^{\infty} x(k)h(n-k)e^{-i2\pi fkT} \tag{3.2}$$



(a) Cylinder 1 disconnected ***(b) Healthy Engine***

Figure 3.1 Evolution of the 3D Parameter Subspace at 1500 rev/min

Cavina et al. [75] applied a similar sliding window technique on the analog recordings of the crankshaft position sensor. After applying time-frequency analysis of the instantaneous engine's angular speed, the presence of misfire was easily detectable. By doing so, all damped torsional vibration of the crankshaft were isolated signaling the presence (or absence) of combustion. To test these assumptions, the researchers used an Alpha Romeo 2L,V6, SI engine and validated the produced results on a Lamborghini V12 engine. For both testing and validation, A Virtual ECU (VECU) was used to control both the ignition and the injection to various cylinders in order to generate misfire at different engine operating conditions. For the V6 engine, this experiment showed that a single-misfire condition induces a torsional damped vibration with a natural frequency of f = 85 Hz. The recorded frequency was constant for different engine speeds and loads. Therefore, STFT was applied around this natural frequency in order to reduce the overall computational complexity. The final joint angle-angular frequency analysis is suitable for detecting misfire conditions when the number of engine cylinders is very high. However, this technique did

not localize the faulty cylinder, detect the presence of double-cylinder misfire, and remain accurate when the road profile is rough.

3.2. Crank-Angle Domain (CAD) Transform

Rotating systems, like internal combustion engines, often create reoccurring periodical phenomena based on their angle of rotation. Fault conditions manifest themselves only in a certain position inside the system, e.g. defective gear or combustion cylinder. The frequency of these events correlates directly with the rotational angle and speed of the system. With changing rotational speeds, it is hard to localise such events using traditional time-based techniques, such as FFT. In the case of the ICE, the speed differs from cycle to cycle even when the engine is at a fixed rpm. These variations are caused by a change in both mechanical and electrical loads as well as the operating temperatures inside the cylinders. For example, the engine's speed is tightly linked to the torque; as this torque fluctuates in the form of power-stroke pulse-train, the speed wavers according to this pulsation. For these reasons, the analysis of the engine's noise and vibration is rarely performed in time domain. Crank Angle Domain (CAD) is used instead to represent the same information with respect to the relative position of the crankshaft of the rotary machine. CAD transform is the mathematical tool that converts the data from time domain to angle domain.

By isolating a complete revolution of an engine, the CAD transform extracts the exact position of specific types of abnormalities, e.g. piston stroke and slap, irregular valve opening/closing, injection/ignition events, and even combustion events like knocking. Added to that, this new domain detects torsional vibrations by simplifying the engine dynamics and the models that control it. CAD domain requires detecting the start and the end of each cycle by sampling the engine synchronization signals, notably crankshaft and camshaft position sensors, at high angular resolutions. These sensors monitor the relationship between the pistons and the valves in the engine and ensure signal synchronization to different engine cycles. Figure 3.2 presents a synchronization

example of vibration data with respect to the Camshaft Identification (CID) signal. Every recorded pulse in the lower graph signals the beginning of a new engine cycle (720°). These cycles are extracted (red dotted lines) and processed further according to the selected FDD strategy.

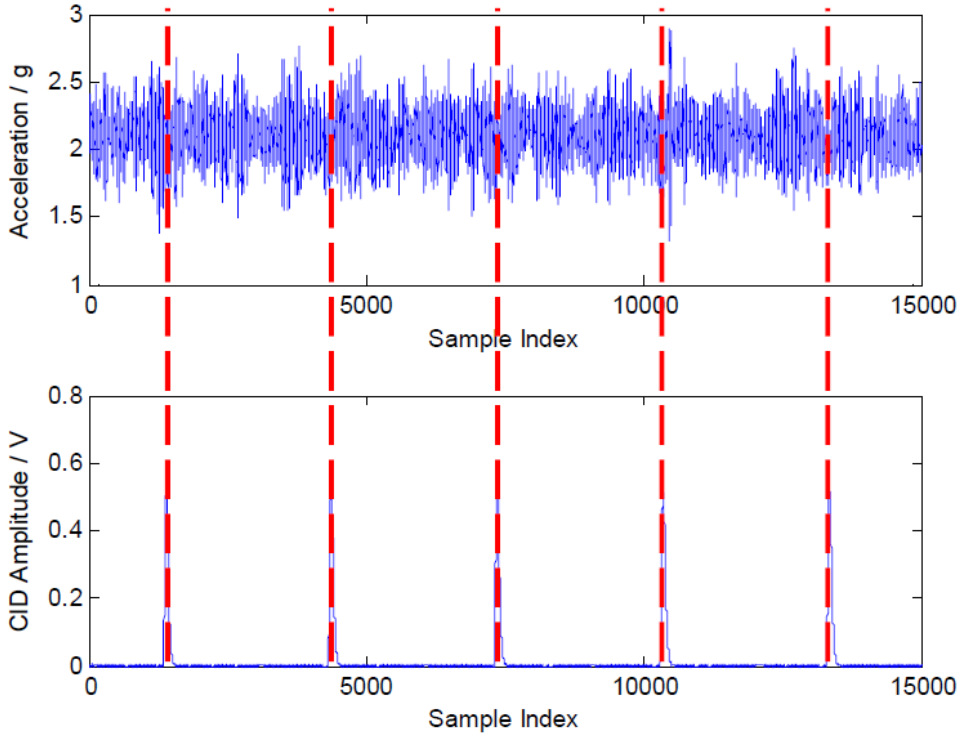


Figure 3.2 ICE-Vibration CAD Synchronization

After detecting the beginning of each engine cycle, both noise and vibration data are reconstructed in CAD through means of interpolation. The interpolation can be linear, cubic spline, or sinc. Arasaratnam and Habibi [76] showed that the reconstruction is more precise using sinc interpolation. Furthermore, they have demonstrated that both truncated sinc, equation (3.3), and windowed sinc, equation (3.4), perform equally for the vibration signal registration problem.

$$sinc_{truncate}(t) = \begin{cases} \frac{\sin(\pi t)}{\pi t}, & t \neq 0 \\ 1, & t = 0 \end{cases} \tag{3.3}$$

$$sinc_{window}(t) = \frac{\cos\left(\frac{\pi\beta t}{p}\right)}{1 - 4\left(\frac{\beta^2 t}{p}\right)} sinc\left(\frac{t}{p}\right) \tag{3.4}$$

In the above equation (3.4), p represents the sampling interval and β is the roll-off factor ($0 \leq \beta < 1$). When $\beta = 0$, the windowed sinc function reduces to a truncated sinc function. With increased β values, the oscillations become shorter in time and smaller in amplitude. Both these functions are used along with the sampled time domain noise and vibration, $x[n]$, to obtain the equivalent CAD signal, $y[n]$, using convolution ($*$), see equation (3.5).

$$y[n] = x[n] * sinc_{fct}[n] = \sum_{k=-\infty}^{\infty} x[k] sinc_{fct}[n-k] \tag{3.5}$$

Despite the well-documented performance of both sinc-interpolation methods (truncate and window), they remain computationally expensive and cannot be implemented online as they require all of past as well as future recordings. For that, this research employs the other types of interpolations, notably linear and cubic spline. The interpolated signal $y[n]$ is the CAD representations of the time domain signal (vibration and noise). This interpolation is a preliminary step that is common among all of the processing techniques covered by this work. And depending on the choice of the FDD technique, this signal can be either used directly to carry the diagnosis or processed further before being used as input to the neural network.

3.3. M1-Average-CAD FDD Techniques

In this FDD technique, the logged time-domain raw data is transformed in Crank-Angle Domain (CAD) by means of linear interpolation. This transformation is followed by a moving-average step

that acts as a filter. Finally the produced average-CAD data is processed further using Fast Fourier Transform (FFT) or Wavelet Transform (WT) to produce three sets of data that act as three separate inputs to the neural network classifier. Figure 3.3 breaks down these steps in details.

As mentioned in section 3.2, after the interpolation step, the obtained CAD data is refined through a moving-average low-pass filter. It is a type of Finite Impulse Response (FIR) filter that removes the unwanted spikes and errors. This processing step is mandatory to obtain a reliable training data set that speeds up the training of the NN and maximizes its accuracy. As the size of the moving-average window gets tuned to the application, the filter smooths the short-term fluctuations due to noisy environment and highlights the long-term trends which ultimately results in a shorter averaged-CAD signals.

There are several types of FIR filters, for example simple, cumulative, weighted, and exponential. The simple moving-average filter, described by equation (3.6), was selected in this work. In equation (3.6), N represents the number of samples, while x_a and x_{a+N} are, respectively, the lower and higher sample points within the averaging window. By fixing the value of N, the filter gets the average of the first N-samples. Then, the filtering window moves one step forward by excluding the first sample x_a and replacing it by the next sample x_{a+N+1}. After traversing the entire dataset, the filter produces the average-CAD data that brings out the dynamics components within the system. Throughout this work, the moving average window size was fixed to 10 samples to produce consistent comparison scheme the different techniques.

$$\bar{x} = \frac{x_a + x_{a+1} + \cdots + x_{a+N}}{N} = \frac{1}{N}\sum_{i=0}^{N} x_{a+i} \tag{3.6}$$

In fast-changing rotating systems, the dominant noise and vibration components manifest themselves in both time and frequency simultaneously. To account for such variations, the transformed averaged-CAD data trains three different neural networks. Figure 3.3 summarizes the M1-Average-CAD FDD technique. This technique relies on three neural networks. The first neural network trains based on the CAD vibro-acoustic data that has been low pass filtered using moving average. The second neural network trains based on the Fast Fourier Transform of the averaged crank-angle domain representation of the vibro-acoustic data. This classification happens solely based on the frequency content of the different engine faults. At last, the third neural network uses the faults' time-frequency features, and it trains using the Wavelet Transform of the averaged-CAD modification of the raw signals. The fixed parametric values used in this technique are presented in Chapter 6.

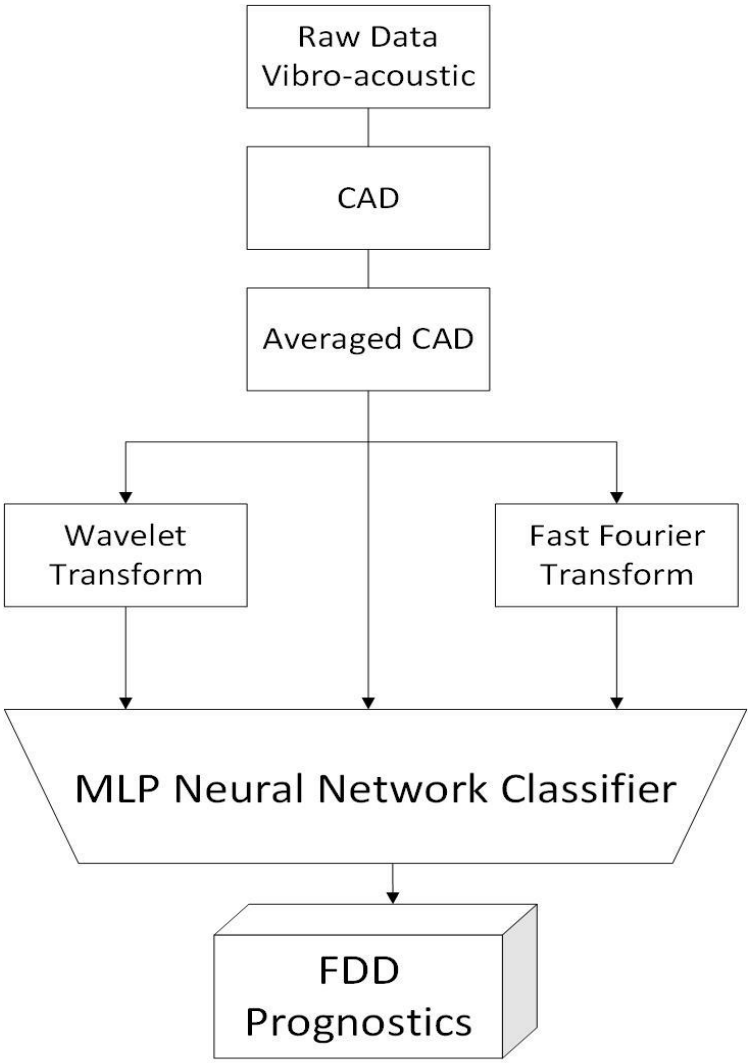


Figure 3.3 M1-Average-CAD FDD Technique Flow Chart

3.4. M2-CAD-IEMSPCA FDD Technique

At the heart of this second FDD scheme lies the Industrial Extended Multi-Scale Principle Component Analysis (IEMSPCA) [9]. What makes IEMSPCA unique and robust is the assembly of two steps:

1- Background noise filtration

2- Extended Multi-Scale Principal Component Analysis (EMSPCA).

These constituting units (EMSPCA + Background Noise Filtration = IEMSPCA) are described in the first two subsections. In this work, IEMSPCA is modified to carry out the feature extraction on the Crank-Angle Domain (CAD) representation of the noise and vibration. The combination of IEMSPCA with CAD is referred to as the M2-CAD-IEMSPCA FDD technique, and it is explained in details in the third subsection.

3.4.1. Back Ground Noise Filtration

Most of FDD applications are located in an industrial environment where the presence of noise can distort the final result of the diagnosis. This distortion is particularly critical when working with both accelerometers and microphones (sound and vibration). One common technique to remove the background noise is to apply noise gating [77]. It is a threshold-technique that allows the input signal to go through, only if the real measurement is higher than the background noise level. When this is not the case, the noise gating technique attenuates the signal by filtering out most prevalent noisy components. To maximize the reliability of such filter, any pre-recorded noise signal is broken into time and frequency. By applying noise gating on both domains simultaneously, the background noise of each individual frequency bin corresponding to each individual time-segment is filtered. Figure 3.4 depicts this process in details.

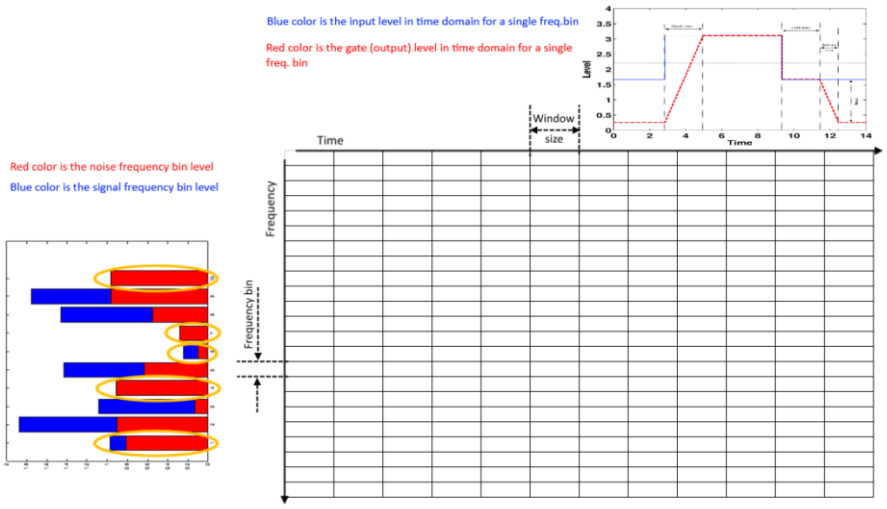


Figure 3.4 Time-Frequency Noise Gating Overview [9]

3.4.2. Extended Multi-Scale Principal Component Analysis (EMSPCA)

The EMSPCA processing algorithm was proposed by Ismail [9], and it responsible for the feature extraction from the engine measurements. According to Figure 3.5, the main difference between EMSPCA and Mod-MSPCA lies in the applied wavelet transform and the placement of the normalization step. While Mod-MSPCA uses the Discrete Wavelet Transform (DWT) that highlights high frequency contents, EMSPCA uses the Wavelet Packet Transform (WPT) which breaks down the frequency spectrum equally. As faults can occur both in high and low frequencies with equal probabilities, EMSPCA is a better feature extraction technique for this FDD application.

(a) Mod-MSPCA

(b) EMSPCA

Figure 3.5 EMSPCA Versus Mod-MSPCA [9]

EMSPCA employs the normalization step at the beginning which improves its generalization capability. By doing so, this technique is less affected by the sensitivity of the sensors, and it produces repeatable results between measurements with different scales and variances due to calibration mismatch. The normalization step is implemented as follows:

$$signal_{normalized} = \frac{signal_{raw} - mean\ value\ (baseline\ center)}{variance\ (baseline\ raw\ signal} \tag{3.7}$$

$$x_{normalized} = \frac{x_{raw} - \frac{1}{n}\sum_1^n x_{raw,i}}{\frac{1}{n-1}\sum_1^n \left(x_{raw,i} - \frac{1}{n}\sum_1^n x_{raw,i}\right)^2} \tag{3.8}$$

As described in Figure 3.5, the normalized signal in processed in time-frequency domain through the Wavelet Packet Transform (WPT). This procedure is carried for both healthy and faulty recordings. Principal Component Analysis uses the WPT coefficients from the healthy recordings as a baseline to measure the intensity of the fault. To do so, PCA extracts the principal component matrices for both healthy $\Sigma_{T,healthy}$ and faulty $\Sigma_{T,faulty}$ measurements. According to equations (3.9) and (3.10), these PCs are generated from both types of measurements by computing the covariance matrix of the principle component score matrix T using equation (3.11).

$$\Sigma_{T,healthy} = COV\left(T_{healthy}\right) \tag{3.9}$$

$$\Sigma_{T,faulty} = COV(T_{faulty}) \tag{3.10}$$

Where

$$COV(T) = E(TT^t) - E(T)E(T)^t \tag{3.11}$$

To be able to generate diagnostics from the results produced by the PCA, Haqshenas [78] used both healthy and faulty principal components matrices ($\Sigma_{T,healthy}\ and\ \Sigma_{T,faulty}$) to define two indices: the first index S_c, which is described by equation (3.12), detects the presence of the fault, and the second index F_c, which is characterized by equation (3.13), isolates the fault.

$$S_{c,j} = \sum_{i=1}^{m} \frac{\sum_{i=1}^{m}\left(\Sigma_{T,faulty} - \Sigma_{T,healthy}\right)_j \circ \left(\Sigma_{T,faulty} - \Sigma_{T,healthy}\right)_j}{\lambda_{i,j}} \tag{3.12}$$

$$F_{c,j} = \left(\sum_{i=1}^{m} COV(F) \circ COV(F)\right)_j \circ \Sigma_{\lambda_j} \tag{3.13}$$

Where

$$COV(F) = P\left(\Sigma_{T,faulty} - \Sigma_{T,healthy}\right)P^T \tag{3.14}$$

In the above equations, the symbol λ_i is the variance of the faulty principle components with respect to the baseline principle components extracted from the healthy measurement. The multiplication ($\circ$) is Hadamard's entry wise multiplication. To use these indices as a fault detection and diagnosis

tool, S_c is first compared against a pre-tuned threshold. If the threshold is exceeded, F_c is computed and compared against a secondary threshold to isolate the detected fault condition.

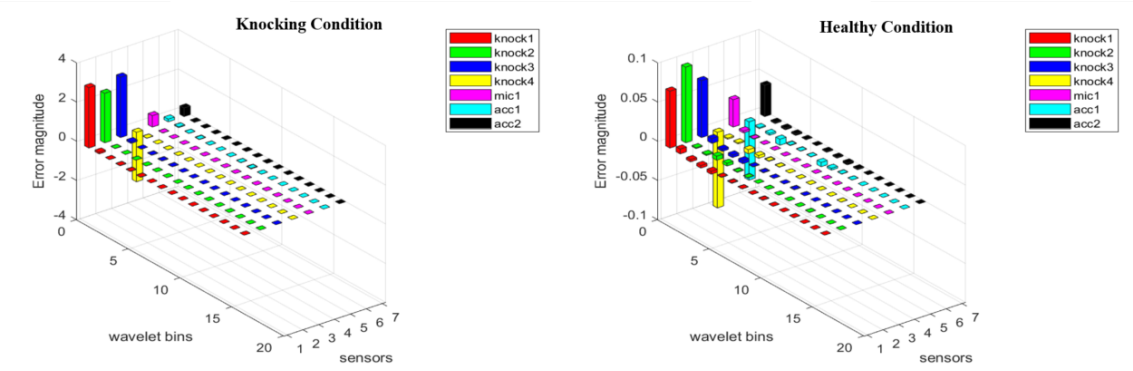
As it is described in equations (3.13) and (3.14), the index F_c contains information that is also represented by the index S_c. To take advantage of this observation, Ismail [9] proposed a combined index, also named F_c, that has a better fault detection capabilities. This defined index ensures the same quantitative representation of faults and limits the number of thresholds involved in the tuning stage. This new F_c index is described by equations (3.15) and (3.16).

$$F_{c,j} = sign\left(L_j\right) \circ \sqrt{|L_j|} \tag{3.15}$$

where

$$L_j = \sum_{i=1}^{m} sign\big(COV(F)\big)_j \circ [COV(F) \circ COV(F)]_j \tag{3.16}$$

The new coefficient F_c takes into account the prior normalization step of the data. Added to that, the presence of the term '$sign\big(COV(F)\big)$' in equation (3.16) allows for negative indices F_c. This feature enhances the classification capabilities of the neural network which uses hyperbolic tangent activation function. With that, the network can classify not only F_c indices that have different amplitudes, but also indices with different polarities. For a diagnostics to be complete, the F_c coefficients are generated for all the sensors.



(a) Knocking Condition ***(b) Healthy Condition***

Figure 3.6 Comparison of Fc Coefficient for Knocking Fault Condition

Figure 3.6 shows an example of computed F_c results for 7 different sensors (4 knock, 2 accelerometers, and one microphone) that were sampled at 200 KHz under moderate knocking condition. Daubechies mother wavelet was selected with a pre-set depth level of 4. Therefore, the spectrum is divided in to $2^4 = 16$ equal in length wavelet bins. By comparing the generated features between the two cases (faulty and healthy), it is noticeable that knocking occurs at lower frequencies for all sampled sensors. Added to that, the recorded amplitudes are very significant for the knocking condition case with F_c amplitudes above unit. These amplitudes were particularly high for the 4 knock sensors which emphasizes their capabilities in detecting knocking fault conditions. This difference between the features is used to train a neural network for later classifications.

Hence, the combination of F_c and S_c into one index allows to identify and isolate fault signatures simultaneously. Added to that, having to generate only one index reduces the amount of the computations and the time required to tune the involved thresholds that would flag the presence of

faults. Generating the combined F_c coefficient represent the feature extraction results of the EMSPCA which is incorporated inside M2-CAD-IEMSPCA.

3.4.3. M2-CAD-IEMSPCA

As discussed earlier, M2-CAD-IEMSPCA technique combines all of the CAD transform and the IEMSPCA algorithm (Background Noise Filtration + EMSPCA). In this second FDD technique of this work, the final step in M2-CAD-IEMSPCA is to use the obtained F_c coefficients to give an intuitive output from the algorithm. For this purpose, a multi-layer perceptron neural network dynamic classifier is applied along with the aforementioned three training algorithms, LM, SVSF, and EKF. The classifier is trained using an extensive dataset of labelled faults, such as misfire, knocking, and pre-ignition. By comparing the new measurement to the trained database, the neural network labels the fault in the new measurement based on its signatures which correlate with the past trainings. To improve both the training speed, and the classification performance of the network, an additional moving average window described by equation (3.6) is applied to the results obtained from EMSPCA. Averaging the features is applicable as they are derived from the same engine cycle. With that, the following Figure 3.7 summarizes the processing steps that make up the M2-CAD-IEMSPCA FDD technique.

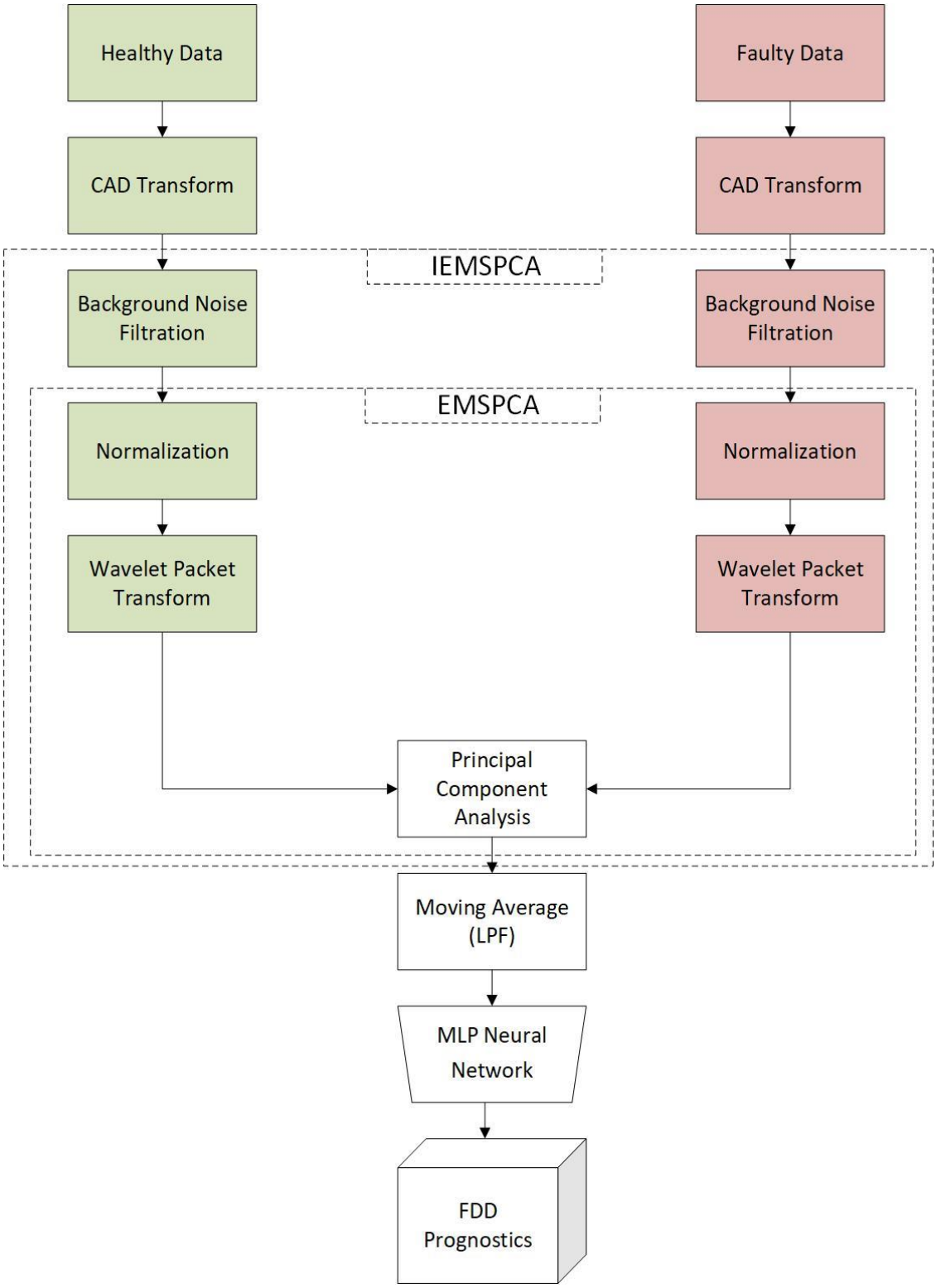


Figure 3.7 M2-CAD-IEMSPCA Algorithm Overview

3.5. Crank-Angle Speed Analysis (M3-CASA) FDD Technique

This FDD technique is introduced by this work, and it focuses particularly on the detection and isolation of engine misfire using crankshaft speed fluctuations. Misfire detection is very important as it directly reflects the characteristics of the combustion torque which is critical for engine control strategies and performance analysis. The misfire diagnostic methods range from relatively simple threshold criteria to sophisticated model-based deconvolution to isolate the contribution of each engine cylinder. Regardless of their differences, these methods are usually very reliable at detecting a specific faulty cylinder at an idle operating condition of the engine. Their reliability and performance generally deteriorate at high speeds, low loads, and rough road profiles.

There is a strong correlation between the produced torque in the combustion chamber and the instantaneous speed of the crankshaft wheel. In fact, at every firing event, a torque pulse is injected to the crankshaft throw. The resulting periodic torque-energy pulses causes the engine and the driveline's angular velocity to fluctuate. When a misfire event happens, the torque applied to the crankshaft changes and results in an altered angular velocity. This shift is mainly due to the lack of positive impulsive torque acting on the crankshaft during the corresponding expansion stroke. Therefore, such events result in the sudden decrease of the engine speed in the analogous pre-defined angular sector. For example, in the case of a 4-cylinder engine with equally spaced combustions of 180°, the sum of the applied torques for a 180°-interval is negligible as the positive torque contributions from the expansion strokes compensate the negative torque contributions from the compression strokes. When misfire violates these normal operating conditions, the resulting sum of the torques becomes negative which causes the crank-angle speed to decrease. These considerations also apply to engines with higher number of cylinders even when the torque contribution of some of the cylinders partially overlap. However, when this is the case, the corresponding angular sector of the misfire event becomes smaller and its effect on the engine's speed becomes negligible.

The Crank-Angle Speed Analysis (M3-CASA) technique requires no prior modelling step, and it relies heavily on the experimental data. According to Figure 3.8, this technique starts by looking at a full engine cycle (720° in CAD) which corresponds to 2 full revolutions of crankshaft wheel. By optimally allocating all combustion events with the corresponding angular interval and by measuring the length of each sector, all single-cylinder contributions can be isolated and their respective angular speeds can be computed using equations (3.17) and (3.18).

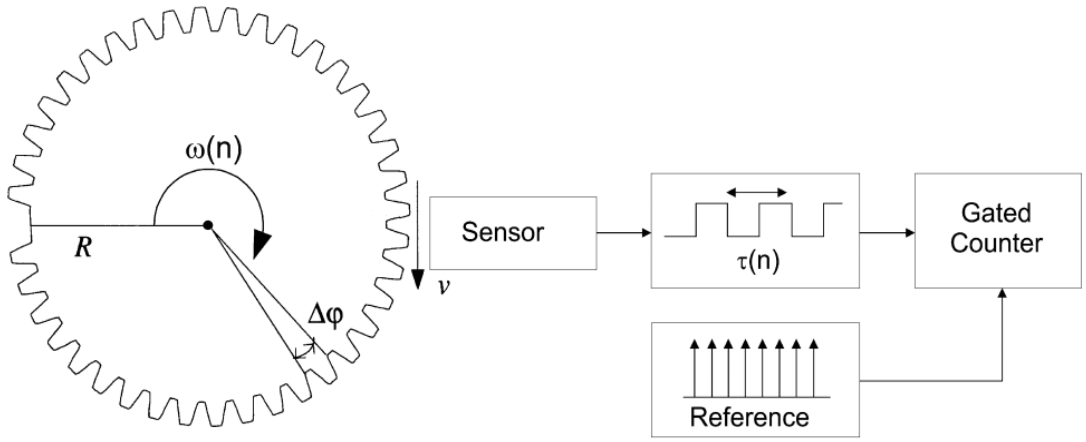


Figure 3.8 Angular Speed Measurement [79]

$$\Delta\varphi = \frac{2\pi}{N} \tag{3.17}$$

$$\omega(n) = \frac{\Delta\varphi}{\tau(n)} \tag{3.18}$$

In the above equations, $\Delta\varphi$ represents the corresponding angular interval that embodies each tooth of the crankshaft. N is the total number of teeth (including one missing tooth at TDC) and is

generally equal to 36. $\tau(n)$ represents the time taken by each tooth to pass in front of the crankshaft position sensor. Finally, $\omega(n)$ reflects the instantaneous speed at each time step n.

After obtaining the angular speeds, the next step computes the difference between consecutive cylinders' speeds and compares the produced results against a threshold. This threshold is fixed for a particular engine speed and load, and it flags the presence of misfire. It is also worth noting that the chosen threshold can be applied in either time or crank-angle domain (CAD) to determine the maximum allowed variation of the velocity over one engine cycle.

The final step in M3-CASA FDD technique differentiates between the two cylinder banks and localizes the specific faulty cylinder by using an additional synchronization signal, such as the camshaft position sensor. This step is required for engine with higher number of cylinders (due to cylinder overlap), e.g. 8-cylinder V-engine. As depicted in Figure 3.9, for the case of ICE, the sensor fusion between camshaft and crankshaft allows to perfectly isolate the faulty cylinders and carry an accurate detection of single-, double-, or even triple-cylinder misfire conditions.

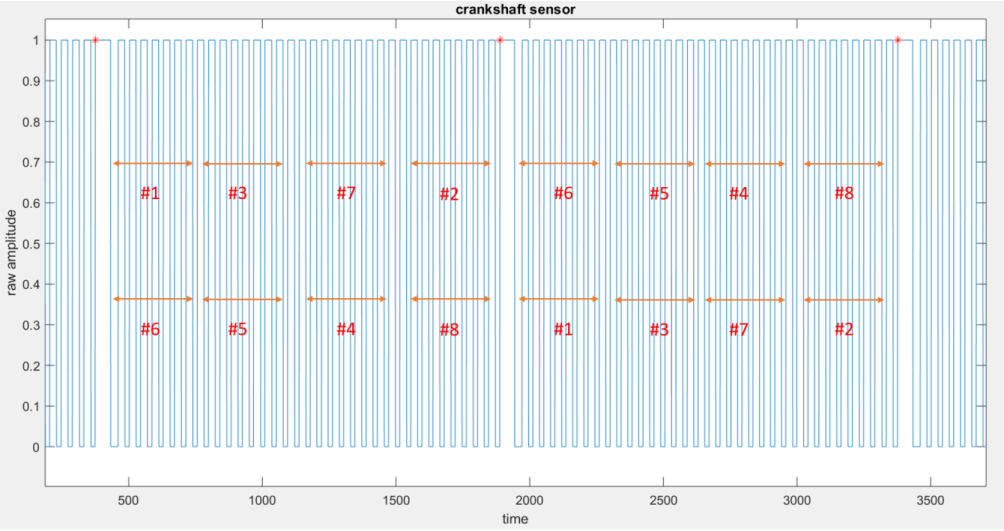


Figure 3.9 Separation of the two Cylinder Bank Using Camshaft Position Sensor

Other techniques that rely on crank-angle speed analysis include pattern recognition using modelled velocity and torque profiles as well as temporal filtering of torque profiles [80]. However, these models are established on limiting assumptions such as: having a rigid crankshaft where excitation of all cylinders is in phase to produce a null cumulative torsional excitation; the ability to temporally filter the composite torque in order to detect individual cylinder contributions; a constant load torque; and a constant clutch stiffness. Additionally, all these methods assume that both stiffness and damping characteristics involved with the driveline are constant. This is not the case in most situations, especially when the vehicle encounters rough road profiles. As such, model-based techniques remain ineffective for misfire detection.

3.6. Summary

This chapter has covered three main Fault Detection and Diagnosis techniques that were implemented in this research as follows:

1- The Average Crank-Angle Domain (M1-Average-CAD) FDD technique that was proposed by Feng [8].

2- The Crank-Angle Domain representation of Industrial Extended Multi-Scale Principal Component Analysis (M2-CAD-IEMSPCA) FDD technique which uses Ismail's [9] IEMSPCA algorithm.

3- Crank-Angle Speed Analysis (M3-CASA) FDD technique which is proposed for misfire fault detection.

All these techniques employ crank-angle domain transformation of the raw data for better synchronization with the engine events. While M1-Average-CAD and M2-CAD-IEMSPCA used computationally expensive tools, e.g. PCA, wavelet, and neural networks. The M3-CASA FDD

technique used a fast algorithm to analyze the engine's speed. Furthermore, this chapter discussed other published work and other potential techniques to diagnose the health of internal combustion engines. The following Figure 3.10 provides a summary of all of the FDD techniques applied in this work.

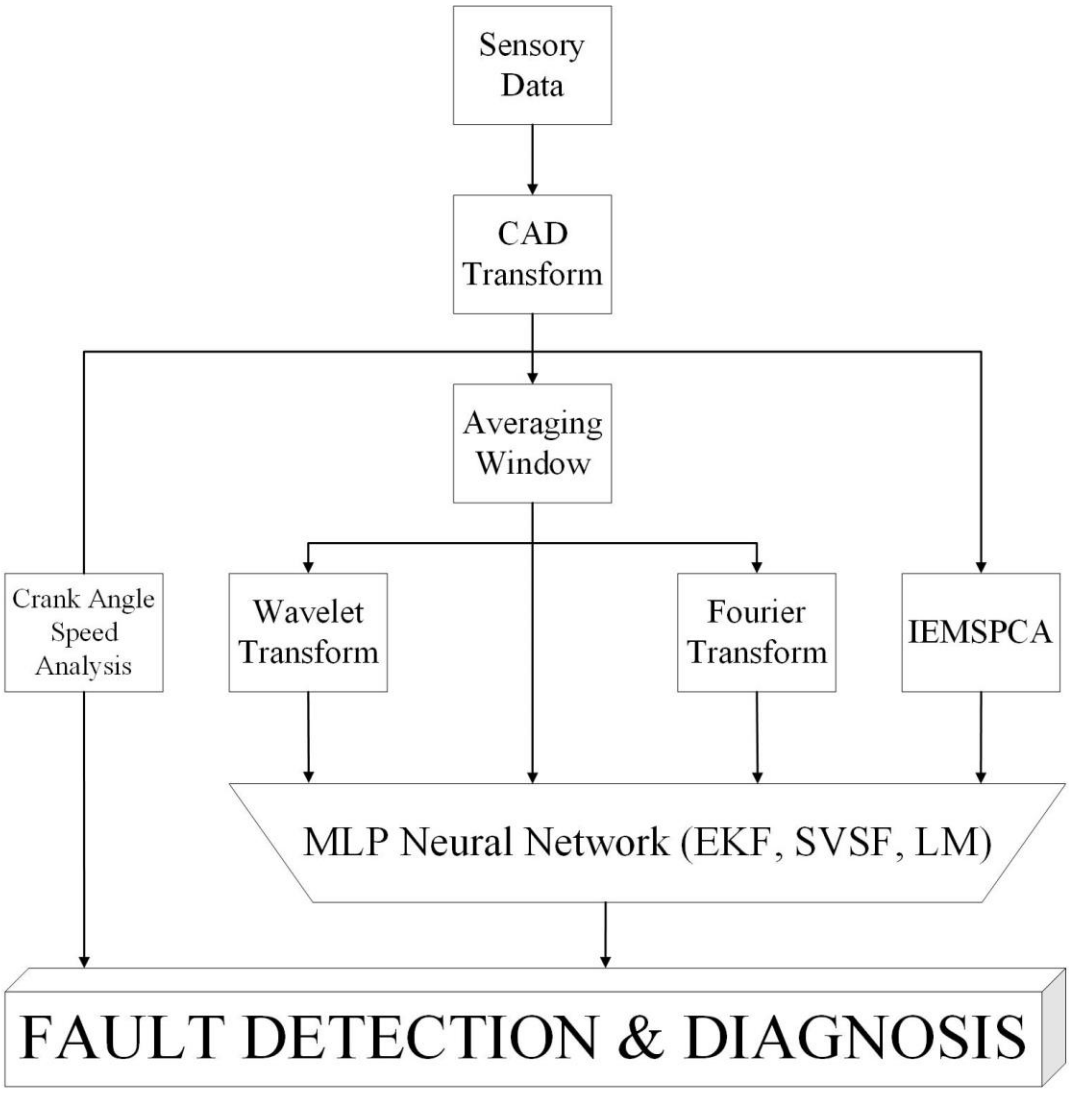


Figure 3.10 Summery of FDD Techniques for ICE

Chapter 4. System Design Requirement Specs

This chapter aims at defining the requirements needed for developing an FDD system that monitors the general health of internal combustion engines. The approach is derived from 'The Engineering Design of Systems Models and Methods' [81]. There are mainly 5 activities associated with this method: developing the operational concept; defining the system boundary; developing an objectives hierarchy; developing the requirements; and, ensuring that the requirements are feasible. The system design is used for developing an FDD system prototype hardware that could run the algorithms described in Chapter 3. This system should have enough computational power to run the FDD algorithms in real-time and have wireless communication capabilities amongst others.

4.1. FDD System Physical Architecture

The FDD system contains both physical and non-physical components. According to Figure 4.1, the non-physical components are summed up under the software umbrella with components such as user manuals, Computer-Aided Design (CAD) tools, device drivers, and operating systems to name a few. The physical components can be separated into 4 main categories.

- **Mechanical Components**: Provide the framework and casing for the system.

- **Electrical Components**: Provide the required circuitry for both signal and power transmission and conditioning. Sensors and interfacing elements are also present.

- **Maintenance Components**: Provide the necessary structural environment, e.g. the lab and a test vehicle, and industrial tool sets, e.g. oscilloscope and power supply, in order to carry a proper maintenance of the system.

- **Safety Components**: Maximise safety overhead to potential perturbations in order to ensure a system that allows for a multi-stage degradation.

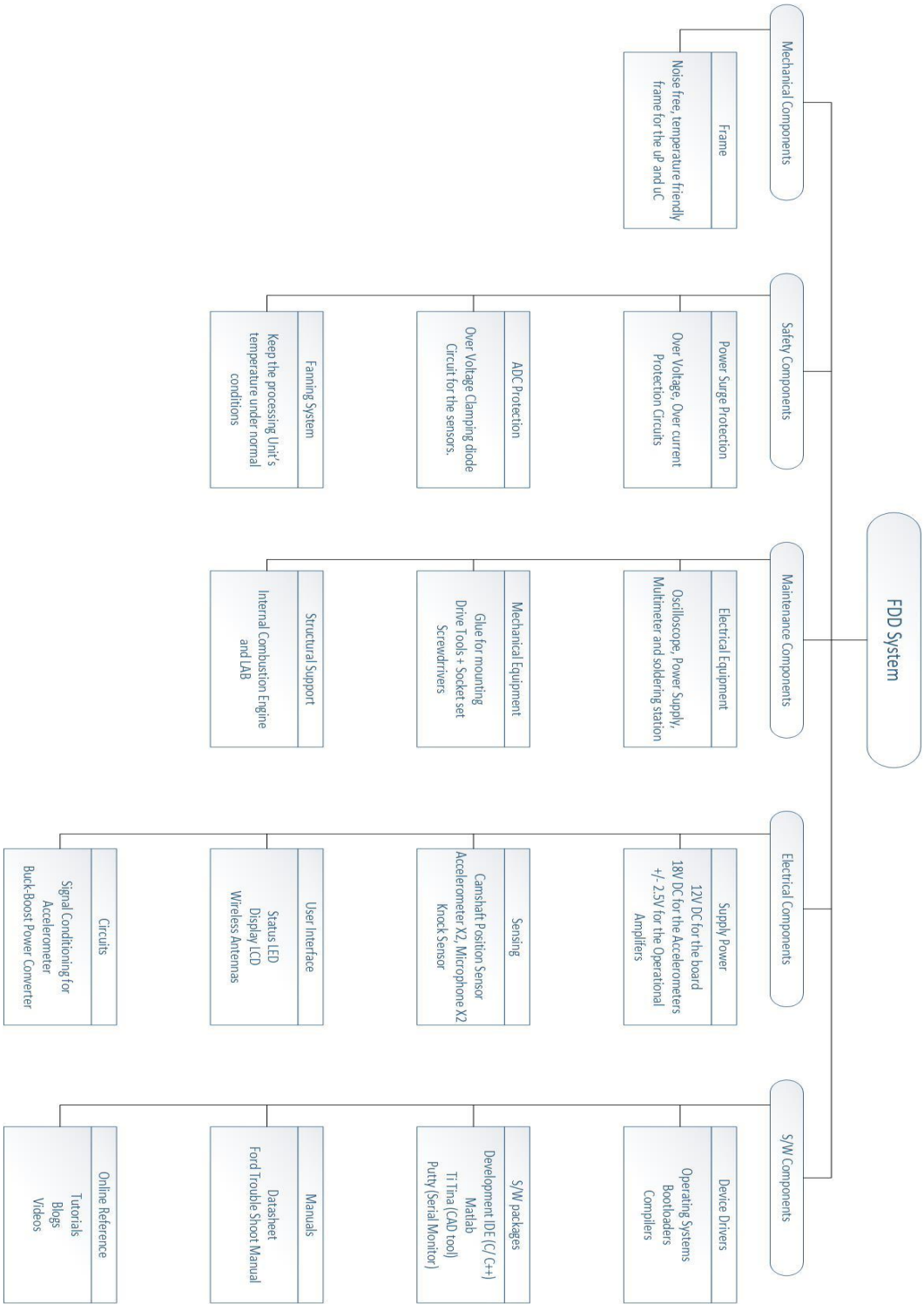


Figure 4.1 FDD Physical Architecture [81]

The FDD system is primarily shaped by its electrical components. These elements are defined based on their added functionalities to the over all system, and they are summed up into 5 principal categories as follows, see Figure 4.2:

- The power conditioning hardware provides the required voltage rails to power the board and its sensors.
- The computation hardware makes the system both faster and smarter.
- The sensing elements enhance the system's perception capabilities of its surrounding.
- The communication features facilitate the usability and the controllability of the device.
- The storage capabilities ensure that all processed data is available for later usage.

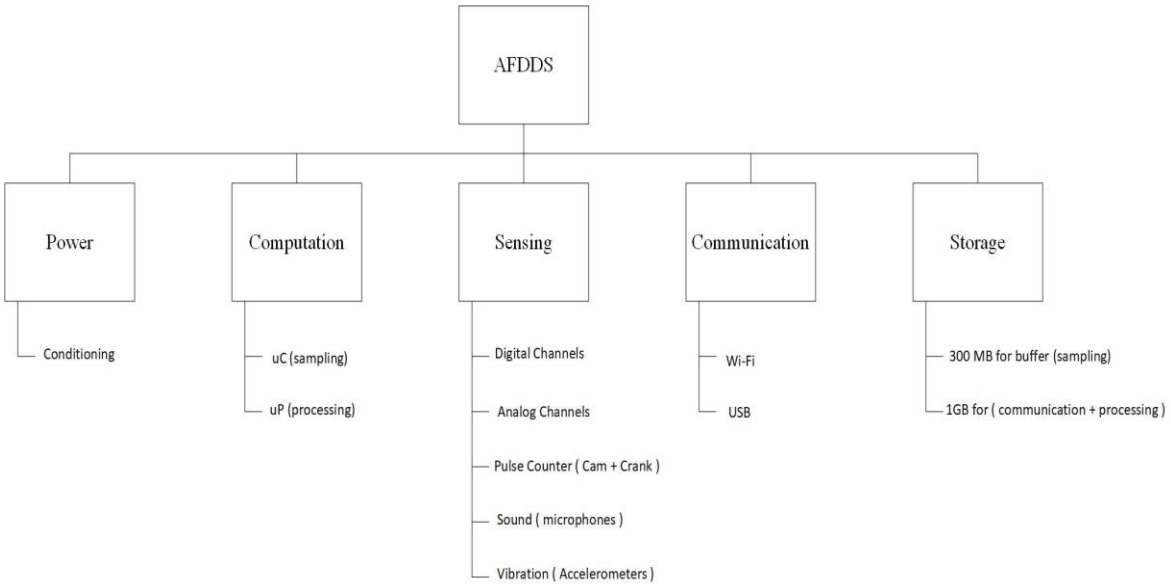


Figure 4.2 FDD Electrical Components [81]

4.2. System Level Functional Design

The FDD system, as the name indicates, is a device that is used solely to carry performance monitoring for Internal Combustion Engines (ICE). The hardware of this device interfaces with sensors some of which are proprietary to the vehicle. The FDD system detects and diagnoses existing engine faults. Its signal processing software isolates and localizes the faults and notifies the user in real-time. In this section, system level design concepts are considered, including:

1. The Operational Concept that introduces the general working theory of the FDD device.

2. The Modes of Operation that describe in details the different functionalities that the FDD device accomplishes once installed in the vehicle.

3. The Operating Environment that coins the limitations and harsh environment that the design of the FDD device should consider.

4. The Objectives Hierarchy that isolates the most important design features to prioritize when building up the FDD device.

4.2.1. Operational Concept

The Fault Detection and Diagnosis system is placed in the vehicle. In doing so, the system has two main external connections with its environment as shown in Figure 4.3. In the first one, the FDD system records all sensors that log the engine's operations. The main sensory components are vibration (knock sensor and accelerometers) and noise (microphones). To synchronize the recorded data with different engine events, both crankshaft and camshaft position sensors are sampled. The FDD system includes a computational platform that is capable of transferring telematics information, fault signatures, and other data such as system device status. The monitoring system

controls the FDD system wirelessly through different queries and commands. Finally, the FDD system power requirements are met by connecting it to the car battery.

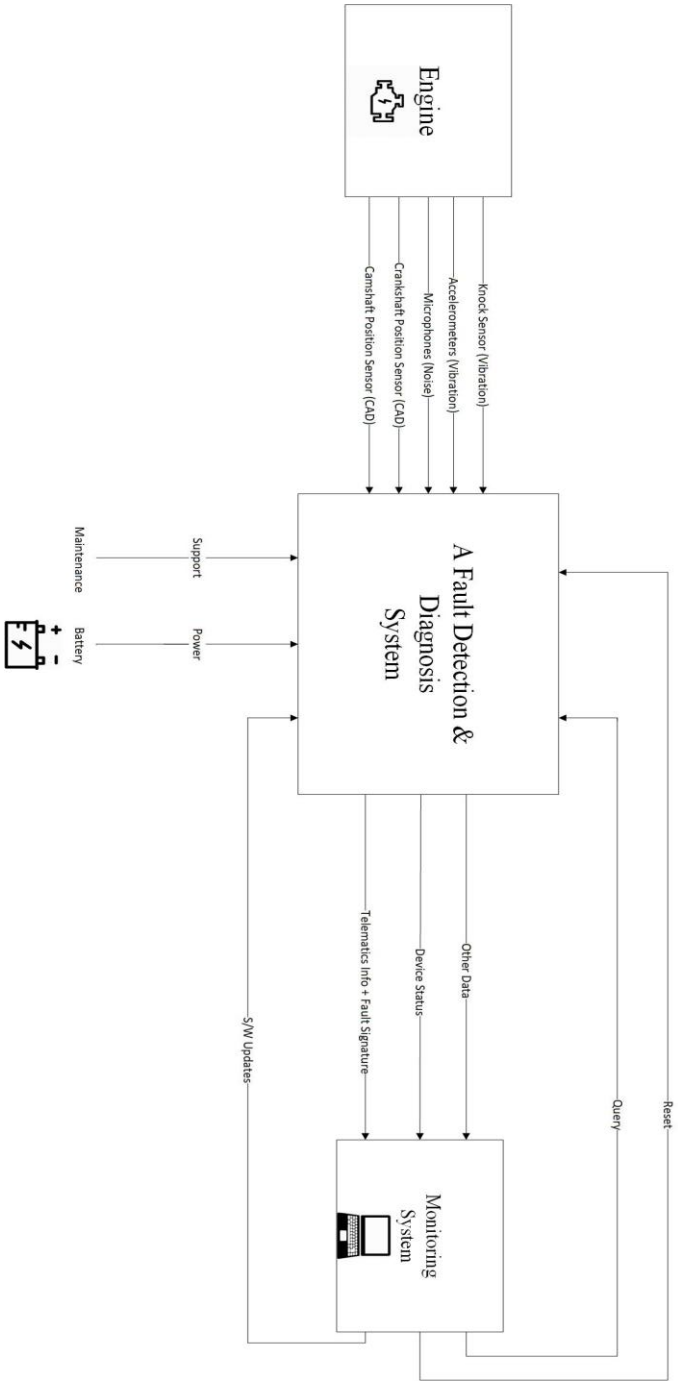


Figure 4.3 The FDD System External Connections [81]

4.2.2. Modes of Operation

When the system is added to the vehicle, it has 9 modes of operations which are described in Table 4.1 . These modes define all possible states that the system takes when operational. When no power is available, the system is in its 'OFF Mode'. Every time the power is provided, the system enters an 'Initialization Mode' where all hardware components are powered and all required communication channels are established. The system exits this mode when recording of the sensory channels has started which puts the system in 'Normal Mode'. Under this mode, the system filters and samples all configured sensory channels and processes the collected raw data. The 'Display Mode' goes hand-in-hand with the previous mode where the system displays the prognostics results to the user. The system has also other modes such as ‘Maintenance Mode’ where both hardware and software updates are undertaken, 'Repair Mode' that allows components' replacement and system reset, 'check Mode' that puts the system in wait state to allow for sensors check, 'Transmission Mode' to move the data from the system to other stations, and a 'Shut Down Mode' to turn off the system. These modes are highlighted within one operational cycle of the device which is described in Appendix A.

OFF Mode	• No Power • No Operation
Initialization Mode	• Powering all H/W Components (Sensors, Circuitry, µC, µP) • Establish the Communication With the System (Wi-Fi) • Start Recording -> Green LED • Start the Wi-Fi Communication • Check for Faults in the System
Normal Mode	• Sample all the Sensors • Carry the Appropriate Signal Conditioning • Carry the A/D Conversion • Fill Buffer (length > 4 engine cycles) • Send the Data to the Main Processor • Process the Data

Display Mode	• Determine the Fault • If Faulty Display the Fault condition else Display Healthy Condition
Maintenance Mode	• Update S/W + H/W • System Fault Detection Report • Manage Sensors' Degradation
Repair Mode	• Replace H/W Components (µP, µC, Sensors) • Update the Firmware • Reset the System
Check Mode	• Check Sensors • Check Communication • Check S/W
Transmission Mode	• Wireless Update of the Data to the Cloud • Wired Update the Data Through USB Cable or a Flash Drive
Shut Down Mode	• Complete Shut Down of the System • No Power -> Keep Saved Data in the Memory

Table 4.1 The FDD system Modes of Operations

4.2.3. Operating Environment

The FDD system will be operating under harsh environmental conditions. In fact, the car engine is known for producing fast temperatures changes (shock) that leads to operating points above 100 degree Celsius [82]. Added to that, constant high pressure, noise, and RF interference affect the quality of recorded sound and vibration [83]. The infiltrated dust, humidity, and solar radiation affect the cooling capabilities of the system [84], while the road profile and engine dynamics takes a toll on the internal connections between the different modules [85].The system is also subject to many other unwanted elements highly delineated as shown in Figure 4.4.

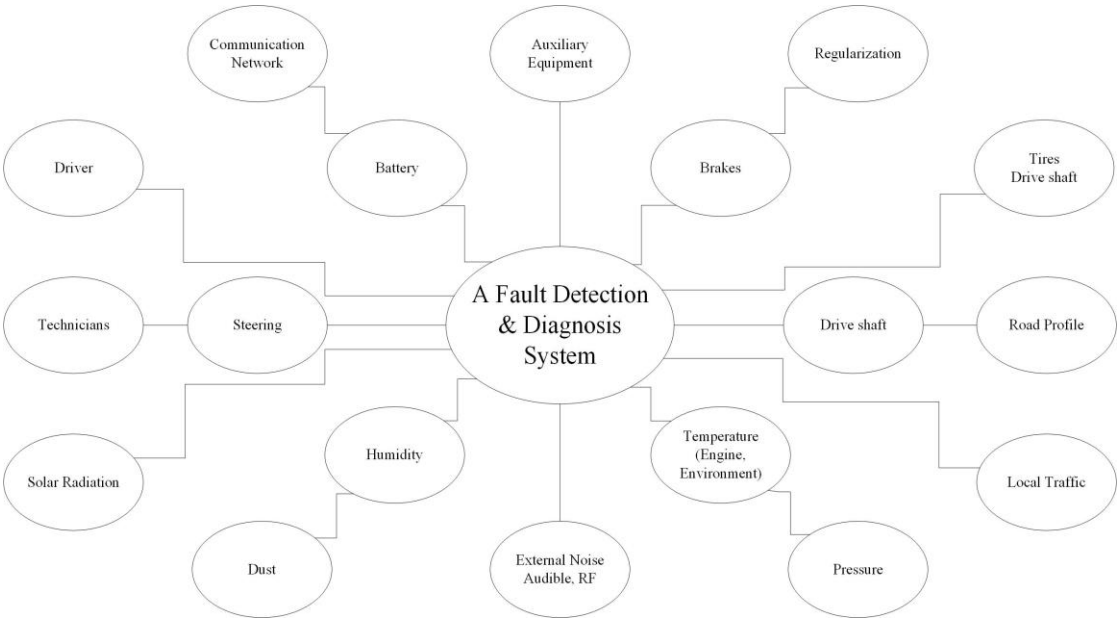


Figure 4.4 The FDD System Uncontrollable [81]

4.2.4. Objectives Hierarchy

To build the first prototype of the FDD device, the main operational objectives are separated into Cost Objectives and Performance Objectives, as shown in Figure 4.5. In order to build a fully performing system that meets the FDD research goals, the cost objectives are highly insignificant (weight = 0.1) compared to the target performances (weight = 0.9). Accordingly, the key player in the FDD system's performance is its processing power and communication capabilities. These two performance criteria are highly needed to achieve the engine diagnosis in real-time using heavy algorithms such as machine learning and wavelet transforms. Furthermore, the final design (Figure 4.6) should be both reliable and flexible to ensure extended normal operations under the hood of the vehicle. Finally, the system should be cheap, power-efficient, and user friendly.

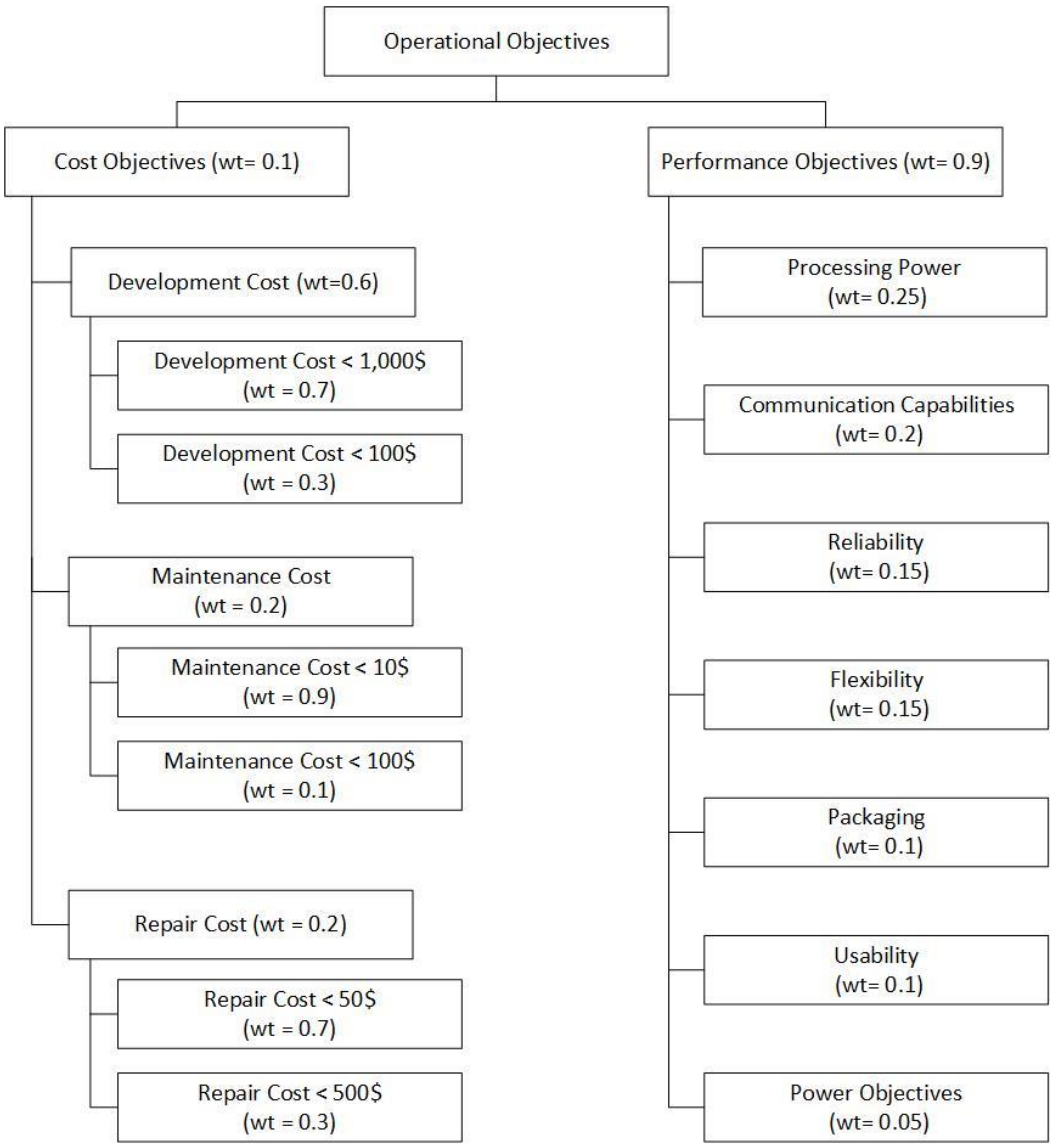


Figure 4.5 Functional Objectives Hierarchy [81]

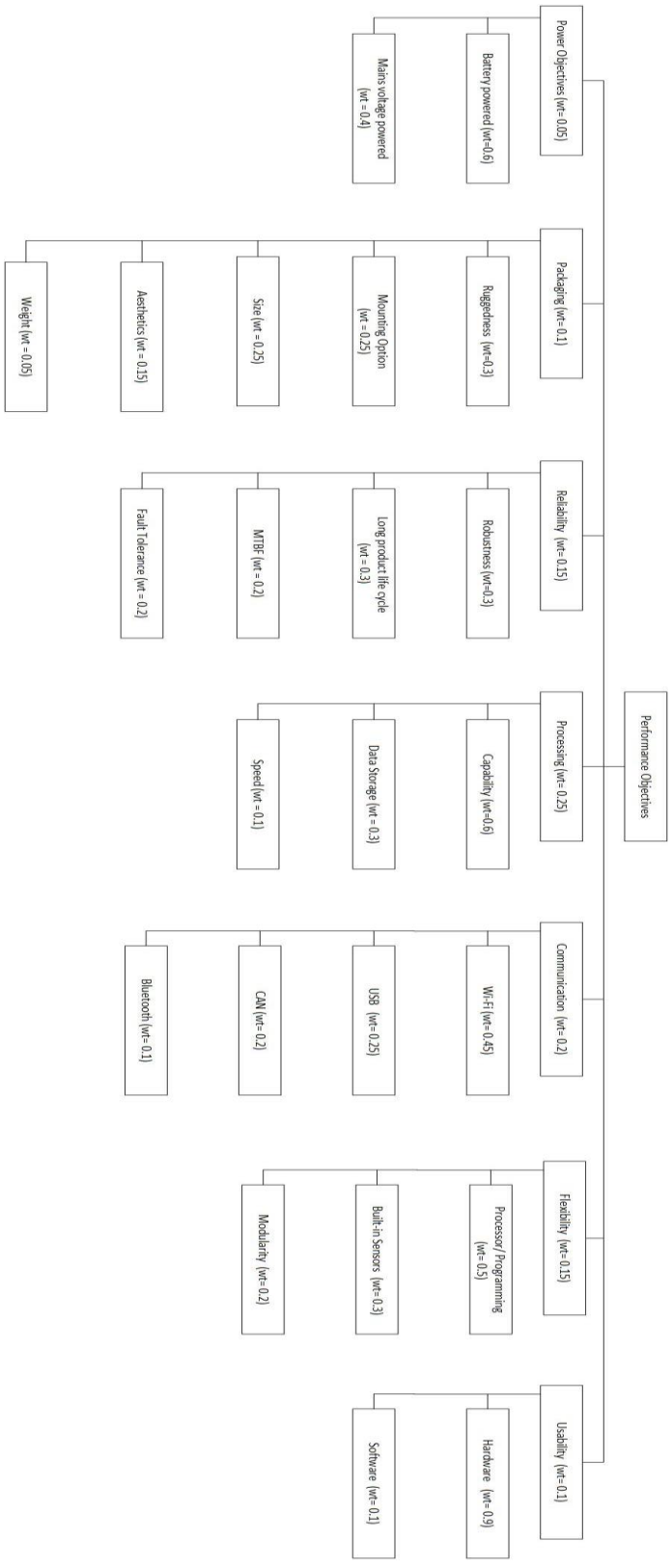


Figure 4.6 Deep Performance Objectives Hierarchy [81]

4.3. System's Functional Architecture

The FDD system, as the name indicates, has the function to provide a comprehensive assessment about the internal combustion engine's health. To do so, it needs sensory data under the right queries and commands to display the fault flags and the diagnosis results.

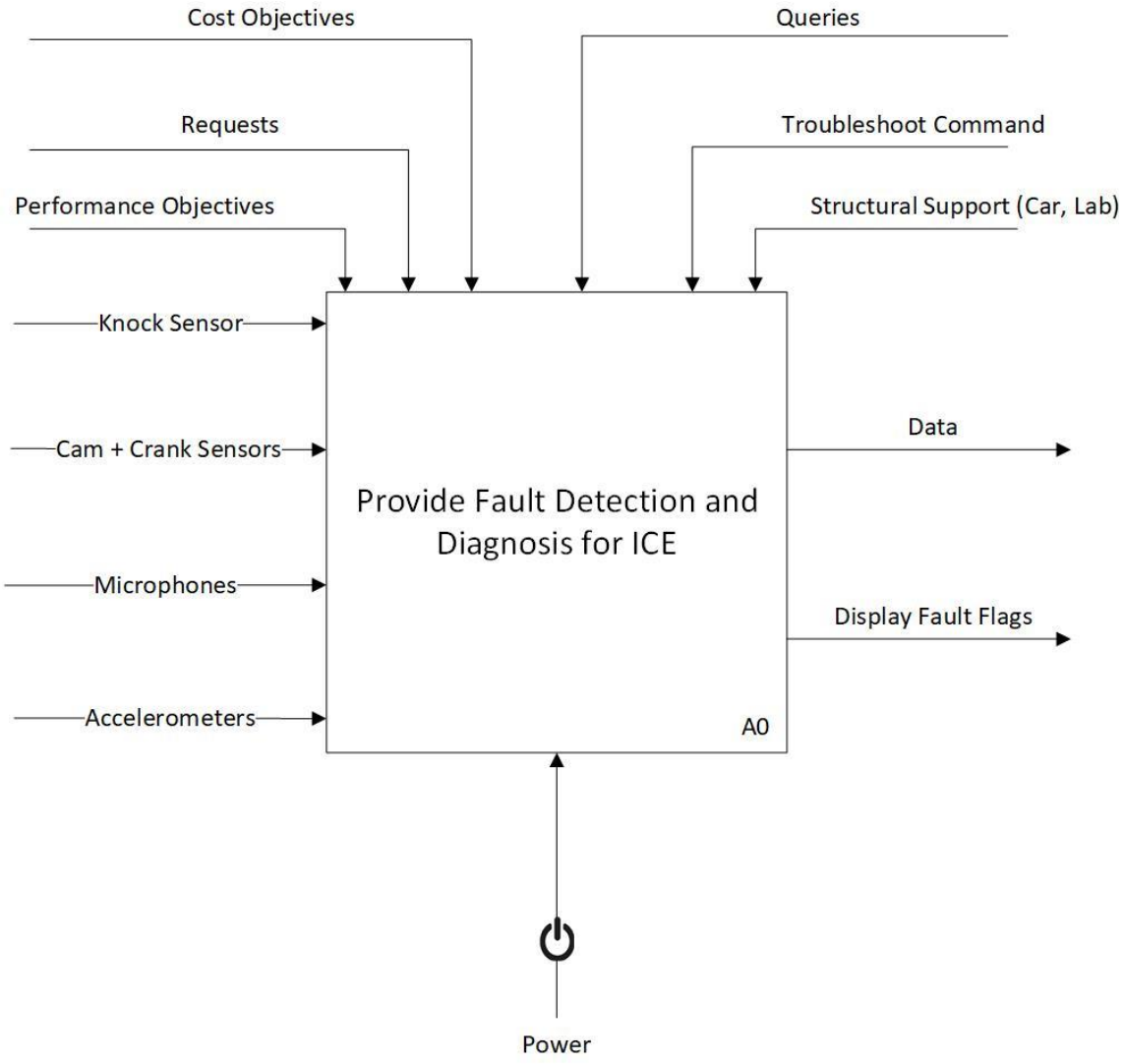


Figure 4.7 Top Level Functional Diagram [81]

On a deeper level, the FDD signal processing technique represent the main function that makes the sensory system unique. To produce viable results, the system samples and synchronizes raw data. The sampling is carried by the A/D module, and the Synchronization is carried by a software module to produce the crank-angle domain representation of the measurements. The signal processing block removes both the noise and the bias. The results of the analysis are communicated to the user. Figure 4.8 depicts these functionalities more in details.

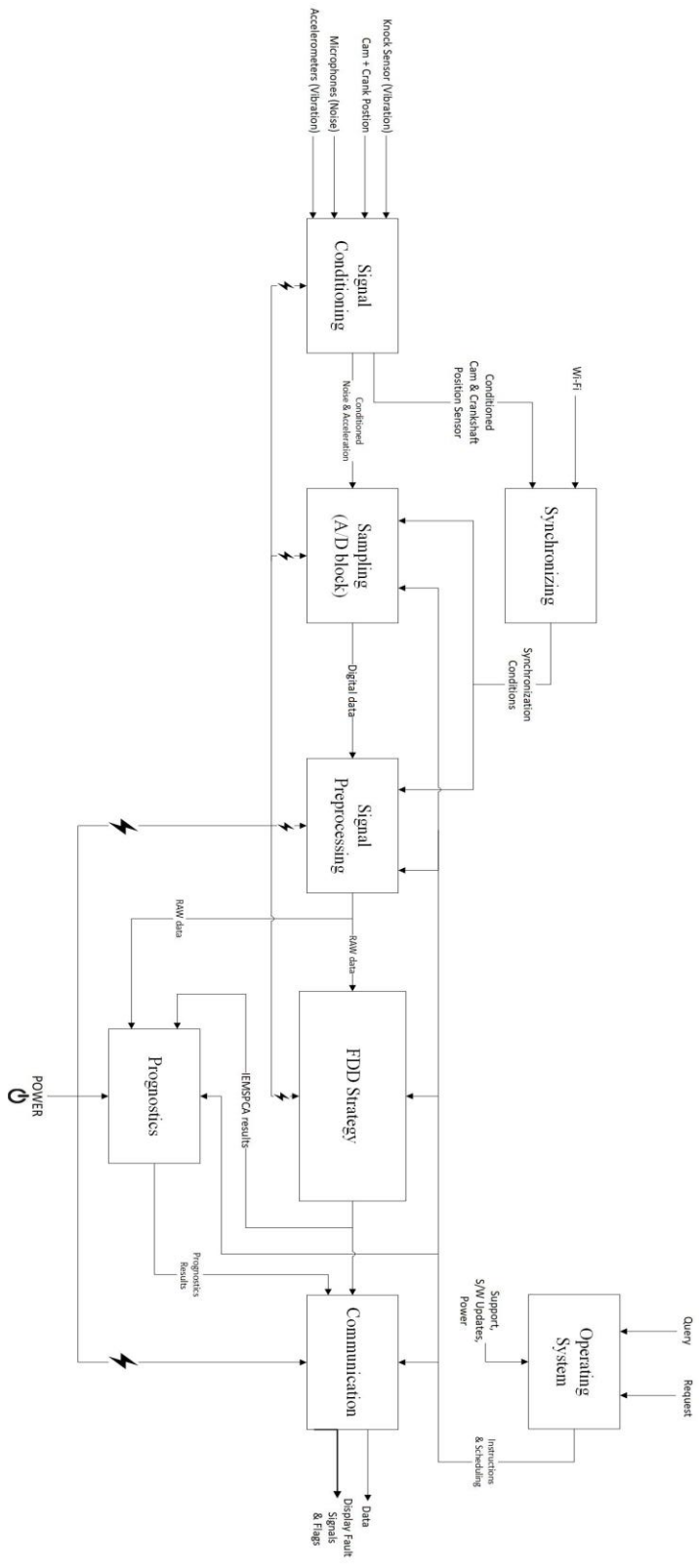


Figure 4.8 Deep Functional Diagram [81]

4.4. System's Operational Architecture

Primarily, The FDD system carries fault detection and diagnosis for the car's engine. To do so, the system has 11 functions accomplished by 11 components. These components are sorted into physical (hardware) or non-physical (software) elements. Some functions such as "buffering" are carried by components belonging to both categories. Figure 4.9 clearly demonstrate that the system relies heavily on two main components: the CPU and the operating system (OS). As a matter of fact, these two components represent the brain power of every embedded application and the FDD system is no exception.

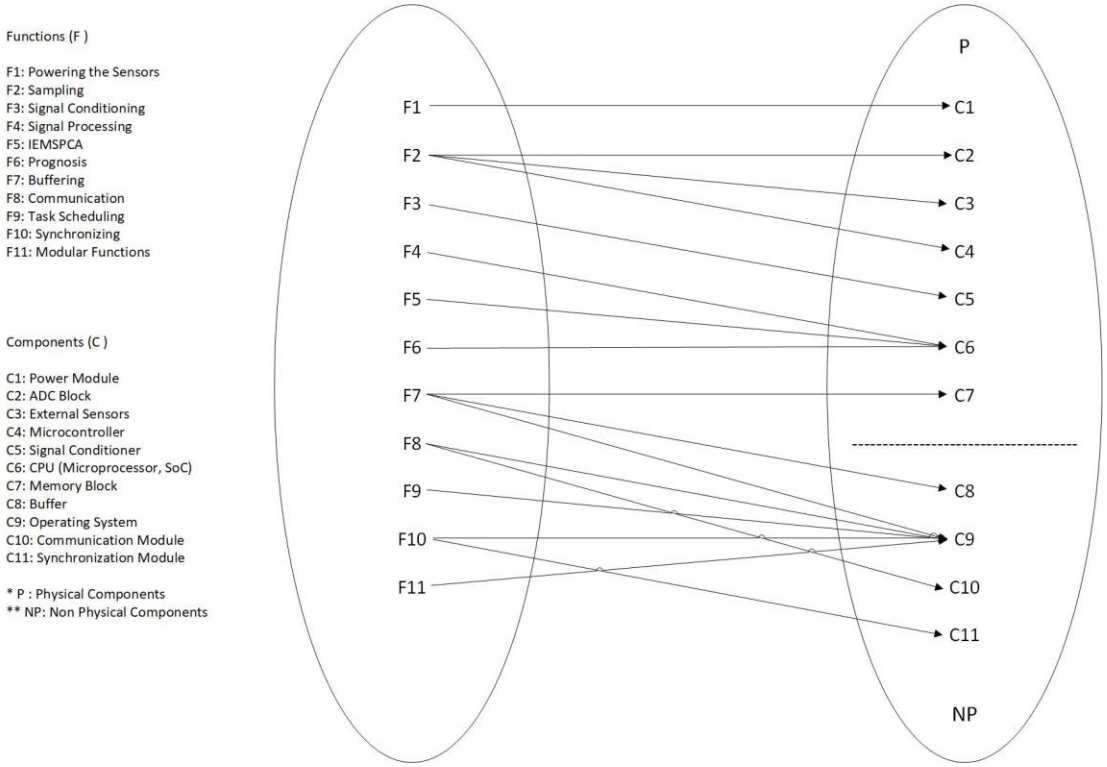


Figure 4.9 FDD Operational Architecture [81]

4.5. Summary

The FDD system is a data acquisition hardware with enough processing power to carry fault detection and diagnosis of internal combustion engine on board of the vehicle. Through 9 modes of operations the system collects the data at rates of at least 40 kHz from several vibro-acoustic sensors. The FDD system processes these measurements and communicates back the prognostics results to the user. After defining the general requirement specifications of the FDD device, the next step is outlined in Chapter 5 and describes a prototype device that meets these requirements. This is followed by extensive testing, analysis, and validation of the target performances.

Chapter 5. The FDD System Design

5.1. Introduction

Since the release of the first data acquisition prototype by IBM, IBM 7700 [86], in 1963 [87], Data Acquisition (DAQ) technologies have evolved to act as the interface between digital computers and the outside world. In fact, DAQs mainly gather signals from sensors and digitize them for storage and later analysis. Depending on the collected data, other functions may include taking a specific action, e.g. sound alarm and light control.

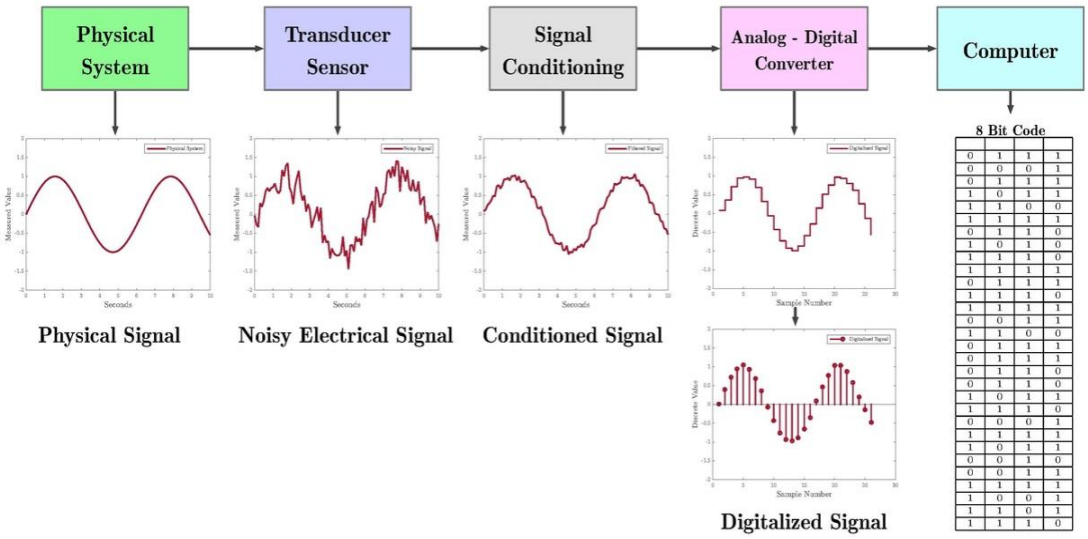


Figure 5.1 DATA Acquisition System Flow Process [88]

According to Figure 5.1, in order to build a custom DAQ system, the following five elements need to be defined:

- Signals of interest to the application: there are mainly two types: analog signals and digital signals. In this FDD application, all of the logged signals are analog, e.g. vibration, sound,

and crankshaft relative position: they are sampled at a frequency of 40 KHz to meet the human hearing range (8 Hz ~ 20 KHz) [89] [90].

- Transducers that detect and convert physical data into electrical signals. The FDD DAQ is capable of interfacing 4 accelerometers, 4 microphones, one camshaft position sensor, and one crankshaft position sensor. The physical properties of all *vibro-acoustic* transducers are selected to ensure a coverage of the aforementioned required-frequency ranges.

- The signal conditioning circuit that filters the recorded data and maximizes the accuracy of the system: this element is critical when dealing with high voltages, noisy environments, and extreme high/low amplitude signal measurements [91]. For the FDD application, the signal conditioning filters the unwanted components in sound and vibration and digitizes the analog signals for both the camshaft and the crankshaft position sensors. In parallel to that, the FDD signal conditioning circuits guarantee all recorded signals' amplitudes to be within the tolerance of the DAQ hardware.

- The DAQ hardware which carries the final step of the signal conversion: the Analog to Digital Converter (ADC or A/D converter), is responsible for converting the analog signal into readable binary codes [92].

- The software that is composed of both drivers (firmware) and application software: the firmware provides a software interface with hardware devices in order to carry the sampling of the environment and the scheduling of different DAQ tasks. Meanwhile, the application software customizes the device for the acquisition application by setting up the right configuration.

DAQ systems are key players in robotics and the industrial internet-of things (IoT). The choice of the DAQ's parameters, e.g. resolution, accuracy, channel count, and speed, are all application

specific. For this reason, a DAQ can be created using different types of technologies, e.g. using the TMS320C31 digital signal processor from Texas Instruments [93] [94], to perform key functions in different applications [95] [96] [97] [98] [99].

This chapter starts by covering the DAQ implementations done by previous CMHT researchers, notably the dSPACE and the CC3200MOD prototypes. After that, it addresses the limitations faced by the fist DAQ prototype based on Intel Curie. Finally, this chapter concludes with the current working implementation which is based on Teensy 3.6 ARM Cortex-M4 processor.

5.2. Previous Data Acquisition Methods

This section covers the previous Data Acquisition (DAQ) systems that were developed within the Centre for Mechatronics and Hybrid Technologies at McMaster University in order to carry Fault Detection and Diagnosis of Internal Combustion Engines. There are mainly two setups: the dSPACE-based and the CC3200MOD-based implementations. The last subsection compares the characteristics of the two designs.

5.2.1. dSPACE-Based Implementation

This subsection covers the work done by Feng [8]. In order to carry fault detection and diagnosis of the internal combustion engine, Feng used a dSPACE (Figure 5.2) to gather the necessary sensory data. dSPACE MicroAutoBox 1401/1501 is a DAQ system [100]. It is based on the IBM PPC 750FX RISC microprocessor [101] that provides clocking speeds of up to 800 MHz.

Figure 5.2 dSPACE MicroAutoBox 1401/1501

The dSPACE contains 16 ADC channels; each A/D channel offers a resolution of 12 bits and an input voltage range between 0 and 5V. Each input can be sampled with speeds up to 20 kHz. The DAQ has also some signal conditioning capabilities that offer overvoltage, overcurrent, and short circuit protections. Furthermore, dSPACE provides extra useful features such as CAN serial bus interface [102] as well as general serial communication protocols, e.g. SPI [103]. Connecting this device with the computer is done through the Local Intermodule (LIN) bus [104] [105]. All the communication involving the dSPACE, whether with the computer or with the sensors, is wired. The final implementation uses mains power to drive all connected hardware components. To identify the available options provided by the dSPACE, the following Figure 5.3 illustrates its detailed functional units.

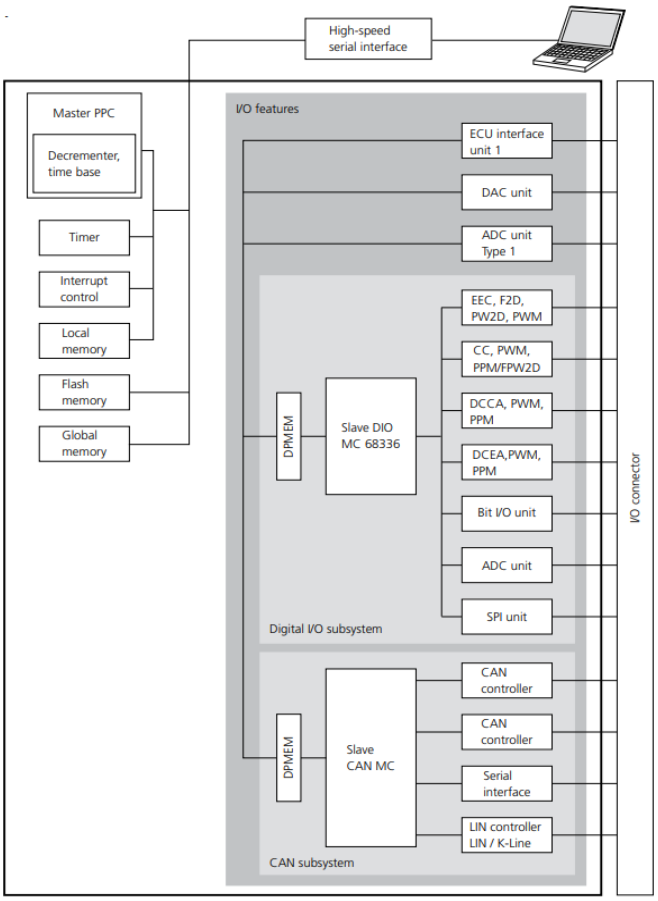


Figure 5.3 MicroAutoBox 1401/1501 Functional Units [106]

In his research, Feng permanently mounted 4 industrial accelerometers directly to the engine block; see Figure 5.4. The accelerometers are the AC240 series [107] from Connection Technology Center Inc. (CTC). The selected sensors have a resonant frequency of 34 kHz, a dynamic range of ±50 g, and a measurement resolution of 100mV/g. These accelerometers require an external 18V (2 x 9V batteries) power supply, CTC PS01 [108], that provides enough current to excite the transducers.

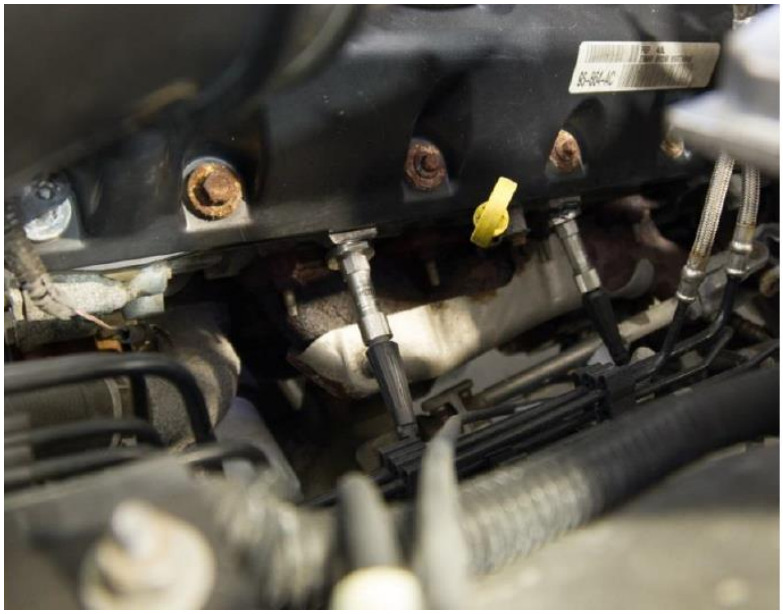


Figure 5.4 Accelerometers Mounted on Engine

To this end, this first DAQ implementation involves a good number of wires. This solution is costly and requires an additional budget to unlock all the features of the hardware as well as maintain the licencing requirements by the dSPACE's software.

5.2.2. CC3200MOD-based Implementation

This subsection covers the work done by Hodgins [7]. In this second attempt, the main component that carries the fault detection is a microcontroller from Texas Instruments Inc., CC3200MOD [109]; see Figure 5.5. This module was selected due to its built-in wireless capabilities. In fact, this microcontroller can handle general internet protocols, e.g. TCP/IP with throughput at 13 Mbps. Finally, this module is based of the ARM Cortex-M4 processor that offers clocking speeds of up to 80MHz.

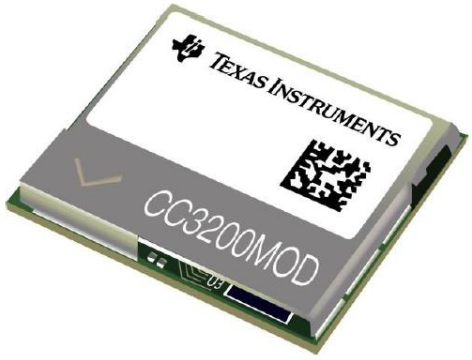


Figure 5.5 CC3200MOD Module [110]

CC3200MOD has 4 ADC channels. Each converter has a 12-bit resolution with an input range between 0 and 1.8V. The device offers a wide range of serial interfaces, e.g. UART [111], SPI, and I^2C [112], which extends its limited capabilities by connecting more modules to it. Furthermore, this microcontroller-based DAQ offers some control features through a 16-bit PWM [113] module and 25 individually programmable GPIO pins [114]. The overall hardware overview is shown in Figure 5.6

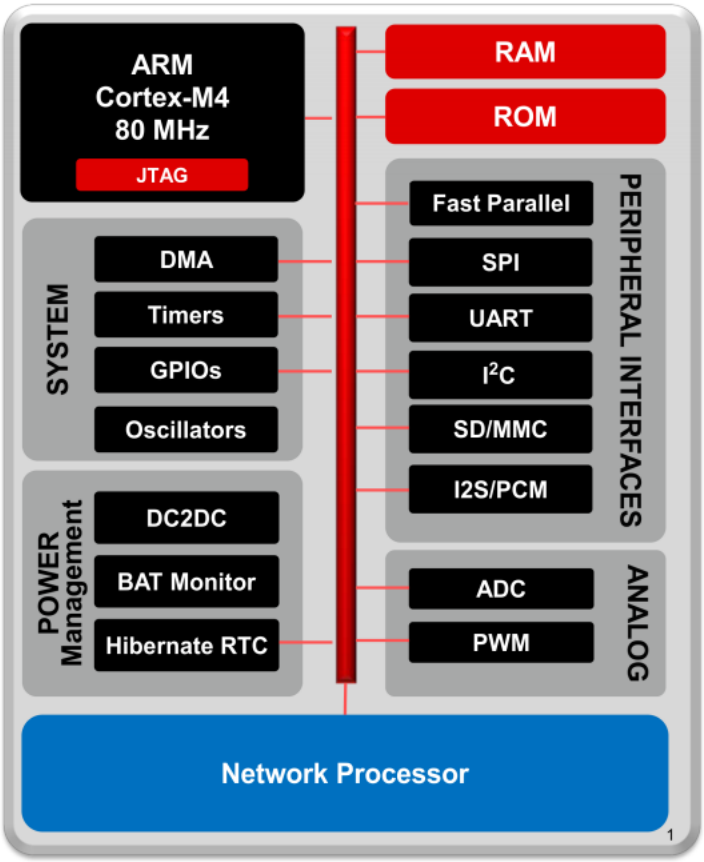


Figure 5.6 CC3200MOD Hardware Overview

To be able to diagnose the internal combustion engine's general health, Hodgins prototyped a Printed Circuit Board (PCB). The final PCB (Figure 5.7) contained a 3-axis accelerometer, 832M1 [115] from TE connectivity, with a ±25g-dynamic range, a 6 kHz-frequency response, and a ±50 mV/g-sensitivity. The board had also a microphone, INMP401 from InvenSense Inc. [116], with its required signal conditioning (amplifier). The microphone has a Signal to Noise Ratio of 62 dBA, a sensitivity of - 42 dBV, and a flat frequency response from 60 Hz to 15 kHz which covers most of the audible range. Another added feature was the incorporation of a temperature sensor to monitor the engine's heat.

Figure 5.7 The Finished PCB Prototype [7]

Unlike the dSPACE specifications, this implementation requires no licence. Users have full access to the running firmware and can modify it accordingly with no additional cost. The communication between the board and the PC can be both wired and wireless. Booting the firmware is done through JTAG port [117] after connecting the PC through CC3200 Launchpad programmer [118]. After booting the necessary firmware, and once the device is sitting in the engine (Figure 5.8), the user receives all the acquired data through the internet. Last but not least, the device is battery-powered where the Li-Po battery [119] [120] is charged using the engine's available energy by means of induction [121] [122].

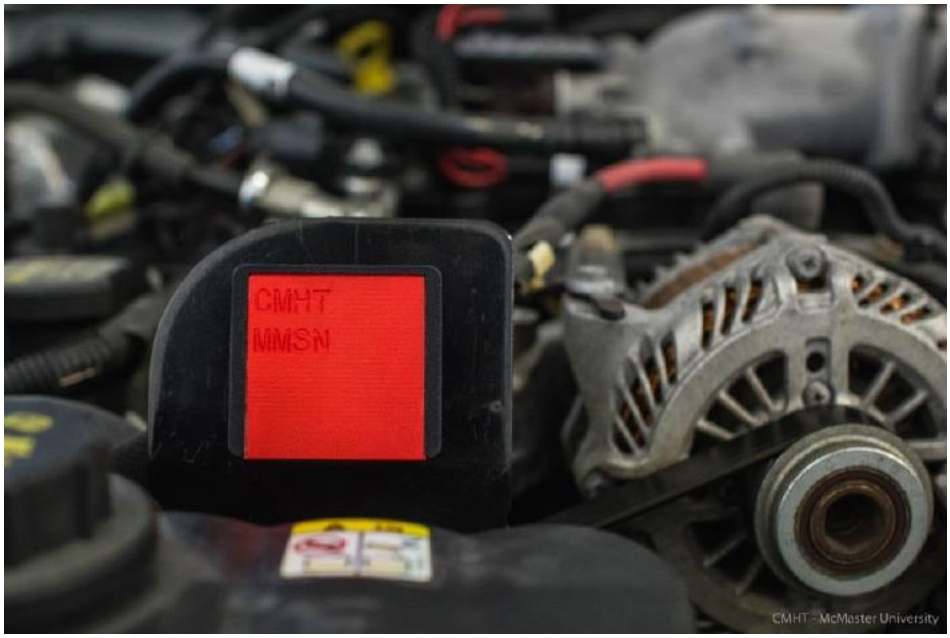


Figure 5.8 FDD Sensor Mounted on the Engine [7]

Downgrading from a microprocessor-based implementation to a microcontroller-based implementation came with a cost. Both the microphone and the accelerometer were sampled individually at 13 kHz which sits below the minimum required sampling frequency of 40 kHz. The addition of wireless capabilities reduced the number of wires hooked to the engine. However, the transmitted data suffers from corruption due to RF interference coming from the engine [83]. Finally, the ICE's extreme temperatures [82] makes any battery powered application redundant.

5.2.3. Comparison of Previous FDD Implementations

The previous sub-sections briefly introduced two working prototypes of fault-monitoring DAQ systems prior to this work. While dSPACE carried the sampling through a microprocessor, the second implementation used a microcontroller. The choice of the processing unit made the two implementations differ from one another. In fact, adopting a specific processor model restricts the number of sensory channels that can be used, the communication capabilities that can be

implemented, and the control features that the final system contains. To illustrate these points clearly, the following Table 5.1 summarizes the main differences between dSPACE and CC3200MOD approaches.

	dSPACE	CC3200MOD
Processor	Microprocessor IBM PPC 750FX RISC (800 MHz)	microcontroller ARM Cortex-M4 (80MHz)
ADC	16 Channels (12 bit, [0, 5V])	4 Channels (12 bit, [0,1.8V])
Sampling Rate	20 kHz Simultaneously	13 kHz Individually
Sampling Type	Sampling by Polling	Sampling by Interrupts
Accelerometer	(X4) 1-axis, 34 kHz, ±50g, ±100mV/g	(X1) 3-axis, 6 kHz, ±25g, ±50mV/g
Microphone	No Microphone	(X1) 60 Hz to 15 kHz
Communication with user	Wired	Wireless
Power supply	Mains power	Battery power
Advantages	Powerful and robust	Compact and open source
Disadvantages	Expensive, too many wires	Not powerful, Short life-cycle

Table 5.1 Summary of Previous FDD Implementations

Both implementations carried fault detection and diagnosis by finding and classifying all 8 cylinder misfire conditions with high accuracy. As indicated in the above table, each DAQ solution leveraged unique assets to carry the ICE diagnosis. This led to the current and final implementation where a working combination of both microprocessor and microcontroller was selected. The final prototype allows for on-board-processing which reduces the overhead coming from data transmission. In doing so, the resulting hybrid solution combines the advantages of both the CC3200MOD and dSPACE implementations.

5.3. FDD System designs

The FDD system design process starts by selecting the hardware components that meet the requirement specifications mentioned in the previous chapter. In addition to that, the observations made about dSPACE and CC3200MOD implementations provide valuable insights about the new data acquisition solution. In fact, the sensory system shall contain both a microcontroller to carry the necessary sampling at the rate of 40 kHz and a powerful microprocessor to run computationally-heavy machine learning algorithms. The final design is a system-on-chip (SoC) that diagnoses the engine's general health automatically.

The UDOO X86 ULTRA [123] presents a suitable solution to the FDD problem. It stands as the most powerful maker board ever created by year 2017. This SoC has a Quad Core 64-bit x86 Braswell 14-nm processor [124]. Its CPU is the Intel® Pentium N3710 that clocks up to 2.56 GHz [124]. What makes this board suitable for neural network applications is the added GPU Intel® HD Graphics (700 MHz) and the fast 8 GB DDR3L dual channel RAM [124]. The UDOO makes a good use of an additional microcontroller that is used to interface the board with its external environment. This microcontroller is the Intel® Curie™ module (Quark SE core 32 MHz plus 32-bit ARC core at 32 MHz) that features not only an additional Bluetooth LE capabilities but also a 6-axis accelerometer/gyro. As seen in Figure 5.9, all these features come in place in one PCB board.

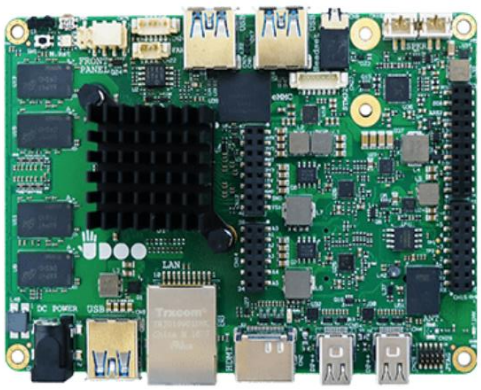


Figure 5.9 UDOO X86 ULTRA [124]

The UDOO X86 ULTRA is a junction point between the PC world, represented by the Braswell microprocessor, and the Arduino world, represented by the Intel Curie microcontroller. This board offers both wired (Ethernet) and wireless (Wi-Fi) internet connectivity allowing for the remote control of the UDOO. When it comes to its wired capabilities, the board offers three USB 3.0 [125] suitable to power any external circuitry connected to it. What's more, the board contains a wide range of serial interfaces, e.g. SPI, I2C, and UART (Figure 5.10) that can be used to extend its capabilities.

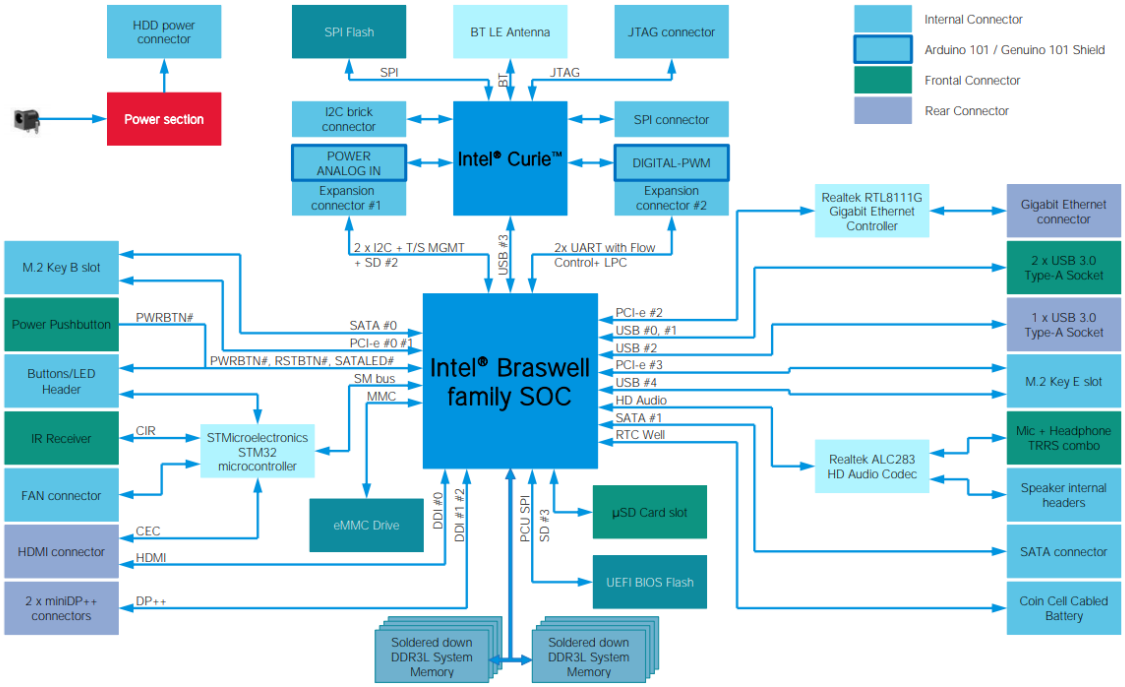


Figure 5.10 UDOO X86 ULTRA Block Diagram [126]

In turning the UDOO X86 into a suitable DAQ for engine diagnostics, two attempts were made. The first one utilizes the existing microcontroller. However, after countless attempts of increasing the sampling rate, the Intel Curie failed to provide a reliable sampling rate at 40 kHz. This failed attempt led to a second solution that integrates the 32-bit 180 MHz ARM Cortex-M4 processor that comes with the Teensy 3.6 board.

5.3.1. First Attempt: The Intel Curie Microcontroller DAQ Implementation

The Hardware Design

This section discusses the supplementary hardware components that make the UDOO board applicable for Fault Detection and Diagnosis of Internal Combustion Engines. These features are summarized into: a main microcontroller, few selected sensors with their appropriate signal conditioning circuits, and some general purpose power electronics components.

The Intel Curie's processor is the Intel® Pentium® x86 ISA with a 32 MHz clock and a 32 bit address bus CPU. The module contains other industry-standard I/O peripherals, e.g. 4 Timers, 4 PWM pins, 2 I2S channels, and 2 UART serial interfaces just to name a few. What is more important, the module avails of 19 channels 12-bit ADC for measurement and 16 GPIOs for control. Figure 5.11 delineates all Intel Curie driving components.

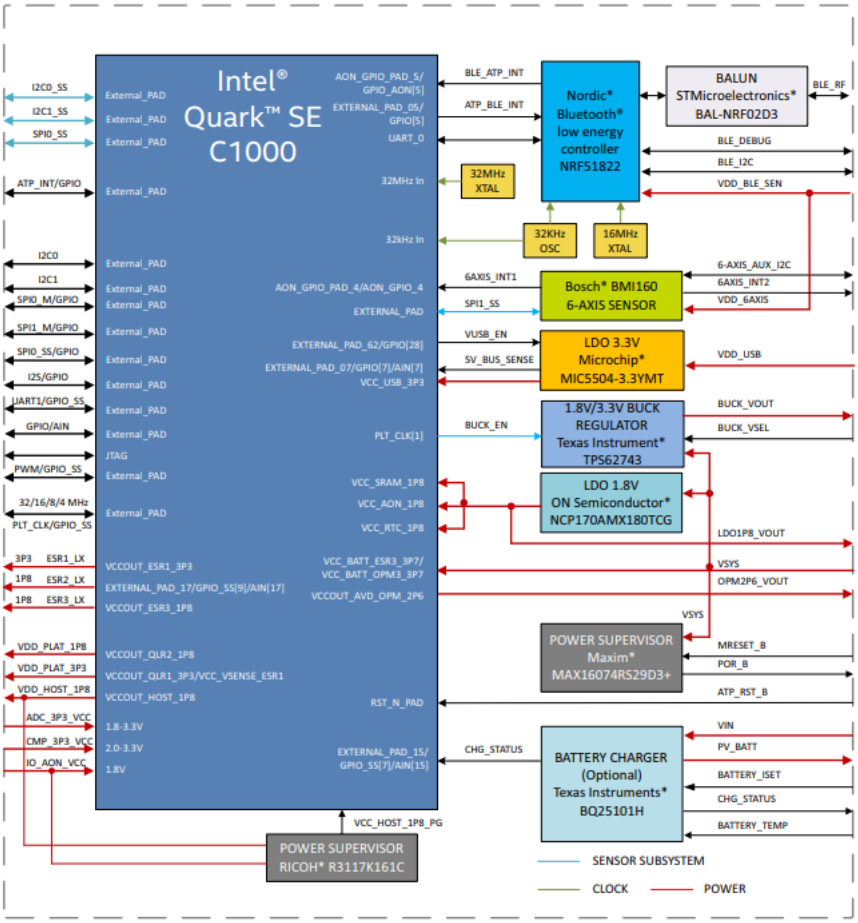


Figure 5.11 Intel Curie Module Block Diagram [127]

According to the above figure, the selected microcontroller chip incorporates a Bosch BMI160 6-axis accelerometer/gyroscope [128]. This sensor communicates with the microcontroller using the general I2C protocol which is slow for the FDD application. To take advantage of the existing industrial sensors, all of the AC240 series accelerometers [107] from Connection Technology Center Inc. (CTC) were recycled. External accelerometers offer more flexibility on the final mounting location when linked to the engine. In order to connect these accelerometers to the ADC of the microcontroller, appropriate signal conditioning was employed. Looking at Figure 5.12, the signal conditioning adjusts the output voltage of the accelerometers from [-10V, 10V] to [0, 3.3V], removes unwanted frequencies outside the bandwidth of interest (1 Hz – 20 kHz), and protects the ADC module from unwanted spikes that may occur during the logging process. Because the general purpose operational amplifier [129], LM 741 from Texas Instruments Inc. [130], was selected, the circuit design and simulation tool TINA-TI [131] was used. The following 3 stage signal conditioning was prototyped.

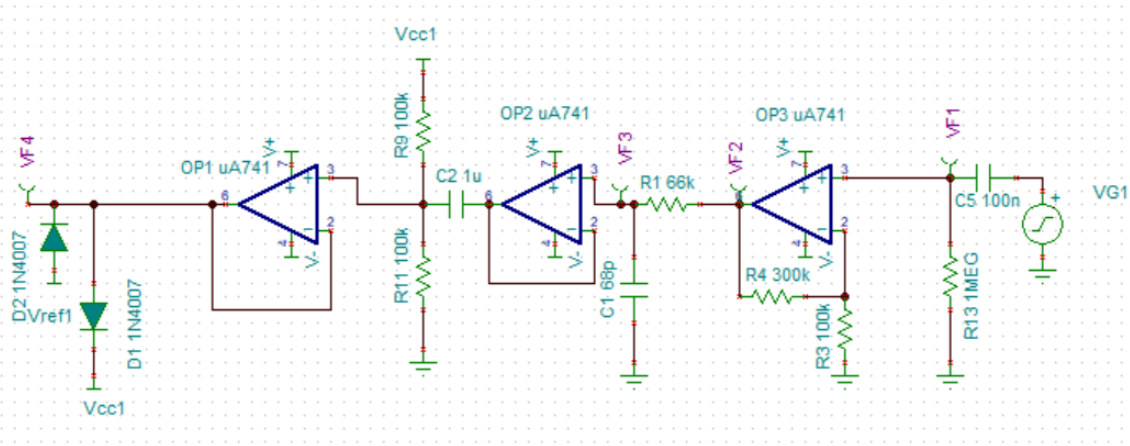


Figure 5.12 Accelerometers Signal Conditioning

From right to left, the signal conditioning starts by filtering the raw vibration using a first order RC High-Pass Filter (HPF) with a cut-off frequency of $F = \frac{1}{2\pi RC} = 1.6\,Hz$ [132]. This removes not only the bias voltage for later amplifications but also the low frequency disturbances coming from the engine's environment. Right after this step, the centred signal undergoes an amplification of a factor of 4 which was calibrated manually using the idle engine's vibrations. Later on, the amplified signal gets filtered one more time using an RC Low-Pass Filter (LPF) with a cut-off frequency of $F = \frac{1}{2\pi RC} = 35\,kHz$ [133] which eliminates all resonant frequencies outside the dynamic range of the accelerometer. Once all filtering is done, an additional bias voltage of $\frac{3.3V}{2} = 1.65V$ centers the vibration wave at the middle point of the ADC's voltage range. The next step involves adding a voltage-follower op-amp circuit to reduce the capacitance of the input signal which allows for faster ADC sampling intervals [134]. Finally, to protect the A/D converters from spikes, two parallel diodes were added as shown in Figure 5.12. The final values of all passive components were selected after testing with different voltage ranges on a breadboard. These tests were first carried using a function generator then using the actual vibration data coming from the engine.

Aside from logging the vibration signals, the UDOO also records the sounds emitted by the running ICE. For that purpose, the MAX4466 Microphone was connected. It is a pre-amp audio evaluation board from Adafruit Inc. [135]. This electret condenser type microphone operates in frequencies between 20 Hz ~ 20 kHz [136]. The sound data gets amplified using the maxim MAX 4466 op-amp [137]. The selected sensor is easy to interface with the microcontroller [138] and provides operating voltages within the limits of UDOO's ADC module.

Figure 5.13 MAX4466 Microphone Pre-Amp Audio Evaluation Board [135]

On top of the accelerometers and microphones, the system logs both the camshaft and the crankshaft position sensors. These recordings ensure synchronization of the sound and vibration with the different engine events. To decrease the data bandwidth, both the crank and cam sensory data were conditioned from their original analog shape to digital binary data. For this modification to happen, a simple inverting comparator Schmitt Trigger [139] was used, see Figure 5.14. Here again, by using the general op amp LM 741 [130], from Texas Instruments Inc., the prototype circuit was simulated using the software package TINA-TI [131]. The output of the Schmitt Trigger goes through a voltage divider with an adjustable factor that guarantees a suitable voltage threshold for the GPIO pin of the microcontroller. This threshold was calibrated and tested manually on a breadboard using recorded data from the engine.

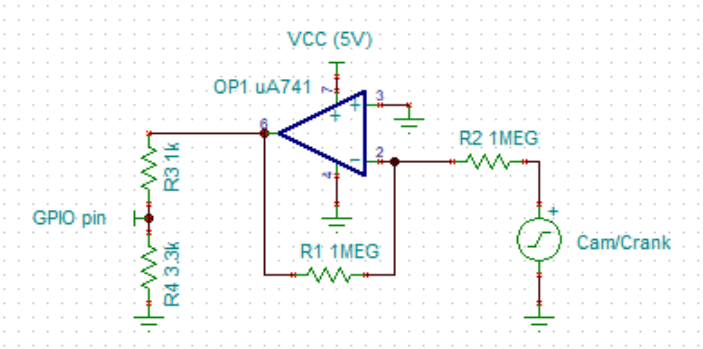


Figure 5.14 Inverting Comparator Schmitt Trigger

The final necessary hardware component is a DC-DC power converter circuit [140] to meet the power needs of the UDOO board. This last requires a constant $+12V_{DC}$, ± 5% and consumes on average 36W to do the basic FDD functionalities. This DAQ prototype is powered from the car's battery which does not provide a constant 12V throughout the different vehicle's modes of operation. In fact, when the engine is at rest, the measured battery voltage at the OBD-2 terminal [141] is around 11.8 V. Once the ignition system is activated, the voltage drops to around 7.8V. Finally, when the engine is running, the recorded voltage sits at 14.3V. Figure 5.15 presents the schematics of the switching buck-boost converter [142] that is used eliminate these variations,

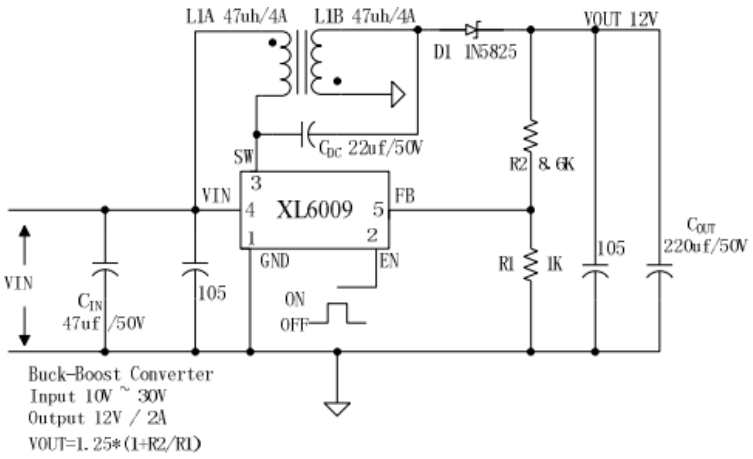


Figure 5.15 XSELMI Switching Buck-Boost DC/DC Converter [142]

With all these aforementioned implementations, Figure 5.16 illustrates the complete hardware design of the DAQ prototype. The next step adds the software elements that make up a fully functioning FDD system.

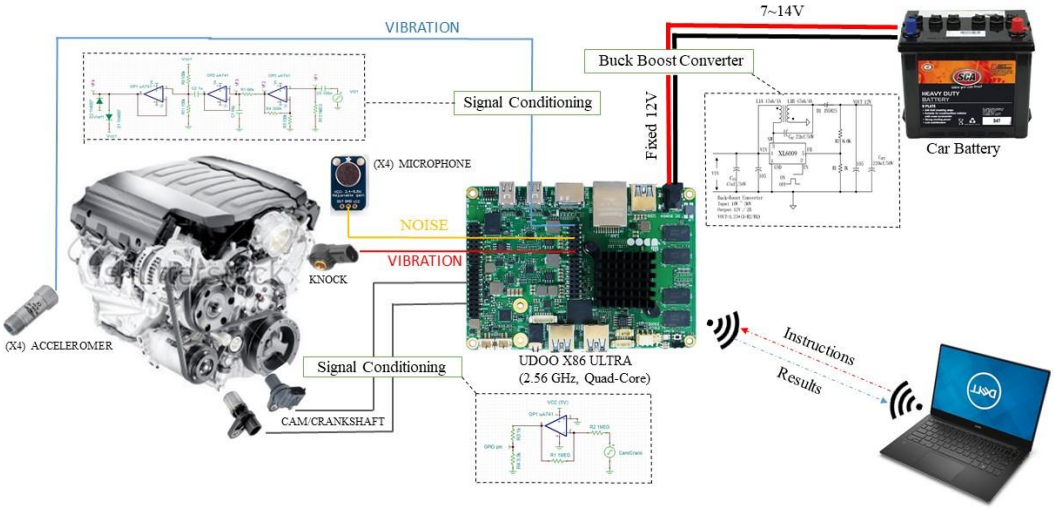


Figure 5.16 FDD System Hardware Components

The Software Design

The software design for the DAQ accommodates both the firmware and the application software. For fault detection and diagnosis purposes, the firmware provides low-level control of the data acquisition hardware components by sampling the sensors, managing the microcontroller's computational resources, and communicating with the host PC (Braswell microprocessor). This section covers mainly design issues faced while developing the firmware of this first implementation. The firmware failed to provide reliable data at 40 kHz. For that reason, the application software, which depends on the appropriate firmware, is investigated in details throughout the second implementation.

The firmware's main function is sampling the sensors connected to the microcontroller. When it comes to sampling procedures, there are two possible implementations. First, sampling by polling [143]. In this approach, the microcontroller constantly checks the ADC block for incoming data. This means, while the ADC is busy carrying the conversion, the microcontroller is locked in a wait-state until the next data is available for collection. For that reason, this sampling scheme is wasteful of the microcontroller capabilities. Added to that, the sampling rate is not constant as the ADC's conversion times are not fixed and depend on many factors, e.g. voltage reference and external capacitance. The solution to this limitation is sampling by interrupts [143]. In this second approach, the interrupt, which is a signal sent from the ADC to the microcontroller, requests the immediate attention from the microcontroller. When that happens, the processor stops performing the current program to service the action required by the interrupt. The taken-response is saved in the callback function of the interrupt, also known as the Interrupt Service Routine (ISR). In this application, the ISR function samples the ADC block and fills the communication buffer with the acquired data. This process repeats until all the sensors are sampled.

According to Figure 5.17 and Appendix B, the sampling scheme starts by the initialization process. This step loads all the libraries needed, initializes all global variables, and configures the parameters that are changed by the user, e.g. the buffer length and the number of sensors connected to the device. After that, the algorithm sets up all of the analog channels, communication channels, and interrupt routines. When entering the main scheduling loop, the firmware turns on the onboard LED. This visual output signals to the user the beginning of the sampling process. Once the FIFO (First In, First Out) buffer is full, the data is entirely transmitted to the main microprocessor. This process repeats until all the analog sensors are sampled during a fixed period of time predefined by the user.

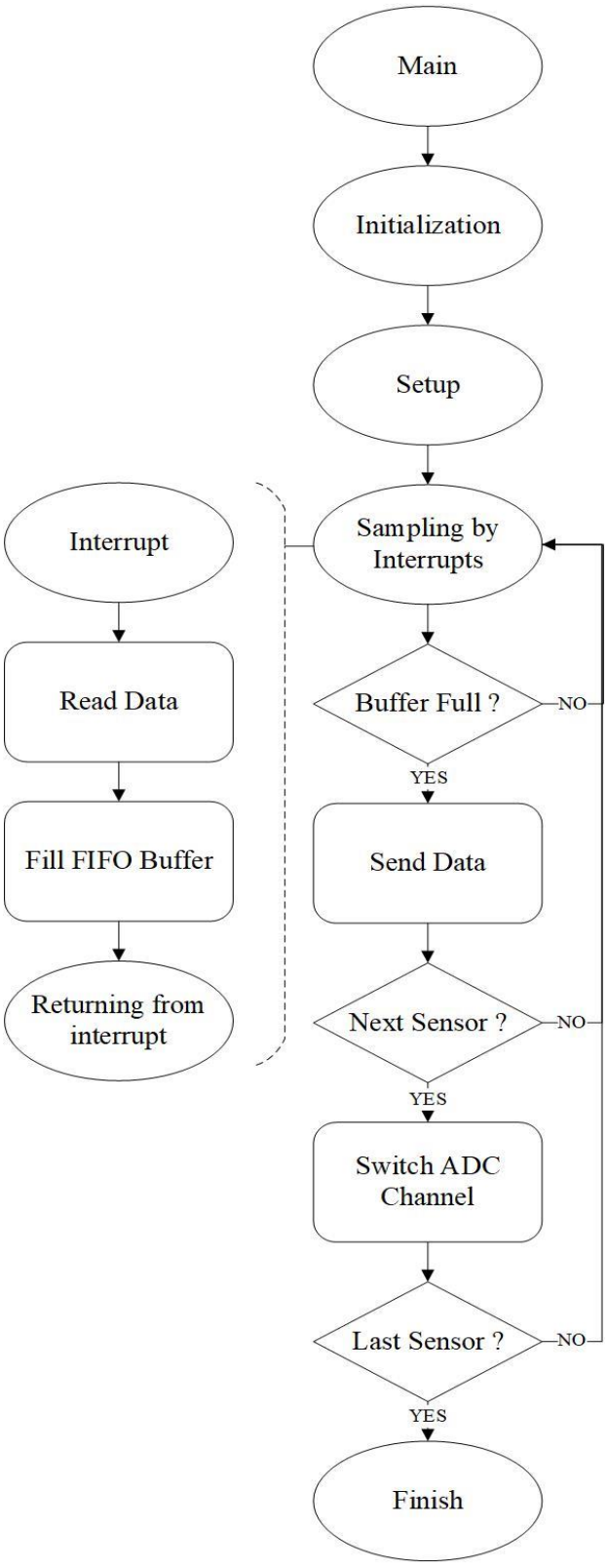


Figure 5.17 Intel Curie Sampling Flow Chart

To have a real-time application, this firmware directly connects with its application software which resides inside the microprocessor. To exchange information between the two chips, the Intel Curie (µC) and the Braswell (µP) connect with one another through an internal USB channel [124]. Additionally, the user needs a serial console to intercept the data packets coming from the microcontroller and save them into a .txt file. For this purpose, Putty Telnet and SSH client[1] were used [144]. Once the sampling of all attached sensors is finished, the user can either sample again to collect more data or process and display the prognostics results using MATLAB software package. The following Figure 5.18 summarizes the FDD system software flow chart. Detailed codes and pseudocodes are included in Appendices B and C.

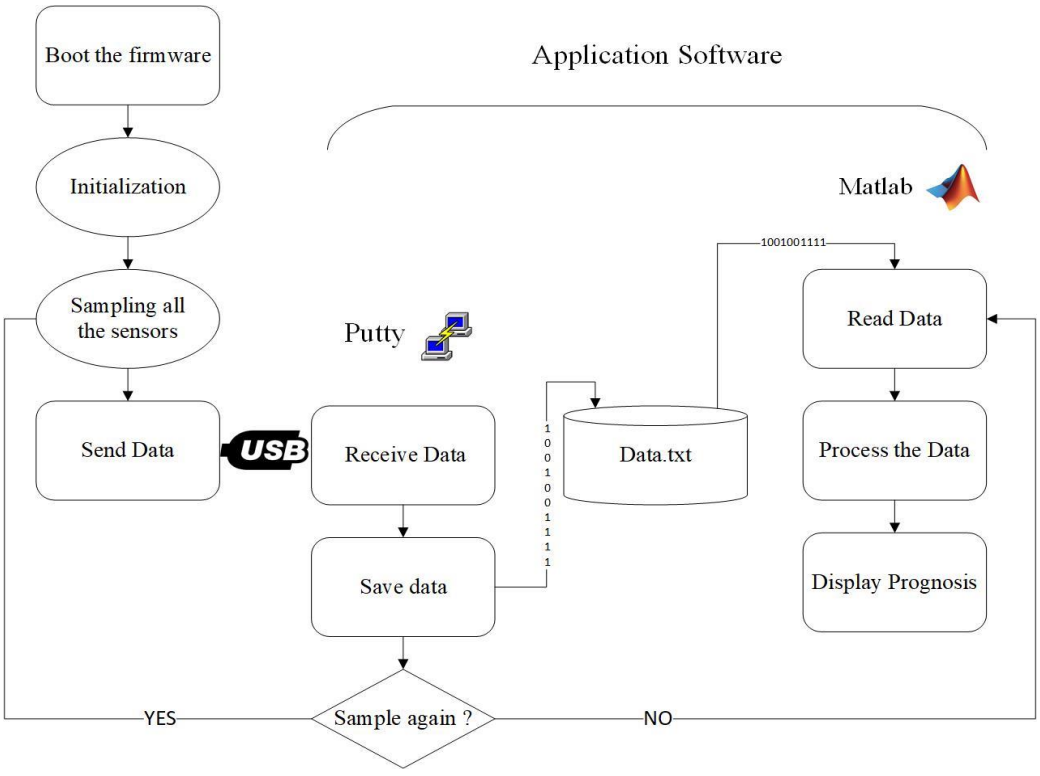


Figure 5.18 The FDD System Software Flow Chart

[1] Putty is an open-source terminal emulator that acts as a serial console for network file transfer applications.

Assuming that the sampling is carried at the right rate, the above flow chart represents a valid data collection scheme. At each cycle the data is collected from one digital channel, either crank or camshaft position sensor, and one analog channel, either knock, accelerometer, or microphone. From one cycle to the other, the tapped analog channel is changed accordingly to log all the sensors.

At the default configuration, the Intel Curie achieves a maximum sampling rate of about 9.8 kHz. This default limit was increased to meet the application needs of 40 kHz after tuning the processor's clock speeds, lowering the delays in the device drivers, adjusting the ADC performances, and reconfiguring the interrupts and communication channels. The below points describe these attempts in details.

- **Increase the clocks' speed**: The quark-C1000 contains three main clocks: a system clock (32MHz), a USB clock (48 MHz), and an RTC clock (32 kHz) [127]. Added to this variety, this chip supports multiple root clock frequencies: a 32MHz high frequency crystal oscillator for highly accurate applications, a 4/8/16/32MHz silicon oscillator for low-power operating mode, and a 32 kHz ultra-low power "doze" oscillator which allows quick wakeup times of the processor. In order for the microcontroller to operate under different clock frequencies, the C1000 supports both Dynamic Frequency Scaling (DFS), to scale the root system and the peripherals clock frequencies, and Dynamic Clock Gating (DCG), to allow for low power operations. According to Figure 5.19, the system's clock can be toggled between 'sys_clk_32mhz' and 'rtc_clk_32khz'.

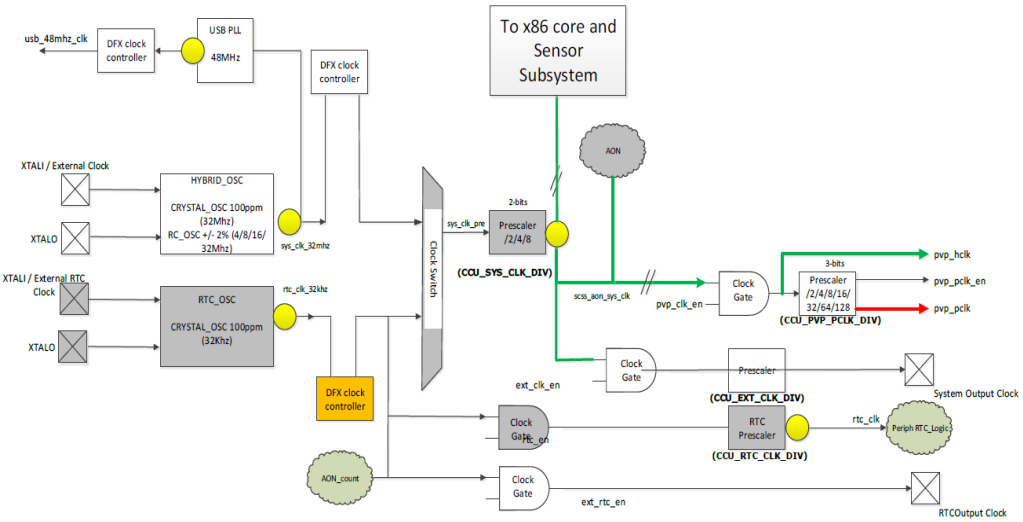


Figure 5.19 Intel Curie General System Clocking Diagram [127]

To increase the system's processing speed, the system clock was set to the maximum of 32 MHz. Before synchronizing the X86 core and sensor subsystems, this clock signal gets scaled through the 2-bit 'CCU_SYS_CLK_DIV' pre-scaler register. To achieve higher sampling bandwidths, CCU_SYS_CLK_DIV was set to lower values (2 or 1). The scaled clock signal is further divided before reaching the ADC block, as shown in Figure 5.20; 'adc_clk' register was set to lower values from its original 128 pre-scaling factor. Before carrying this configuration, the divisor must be selected such that the 45:55 duty cycle of the ADC clock is not violated. Carrying these changes requires updating the flash before the actual clock frequency is changed. Increasing the speed of the processor worsens the ADC performance as the conversion cycles do not have enough time to be completed. In other words, a faster ADC conversion times translate in lower effective number of bits (ENOB) [145] that can be used to represent the data. For that reason, tuning these parameters was done carefully according to the application at hand.

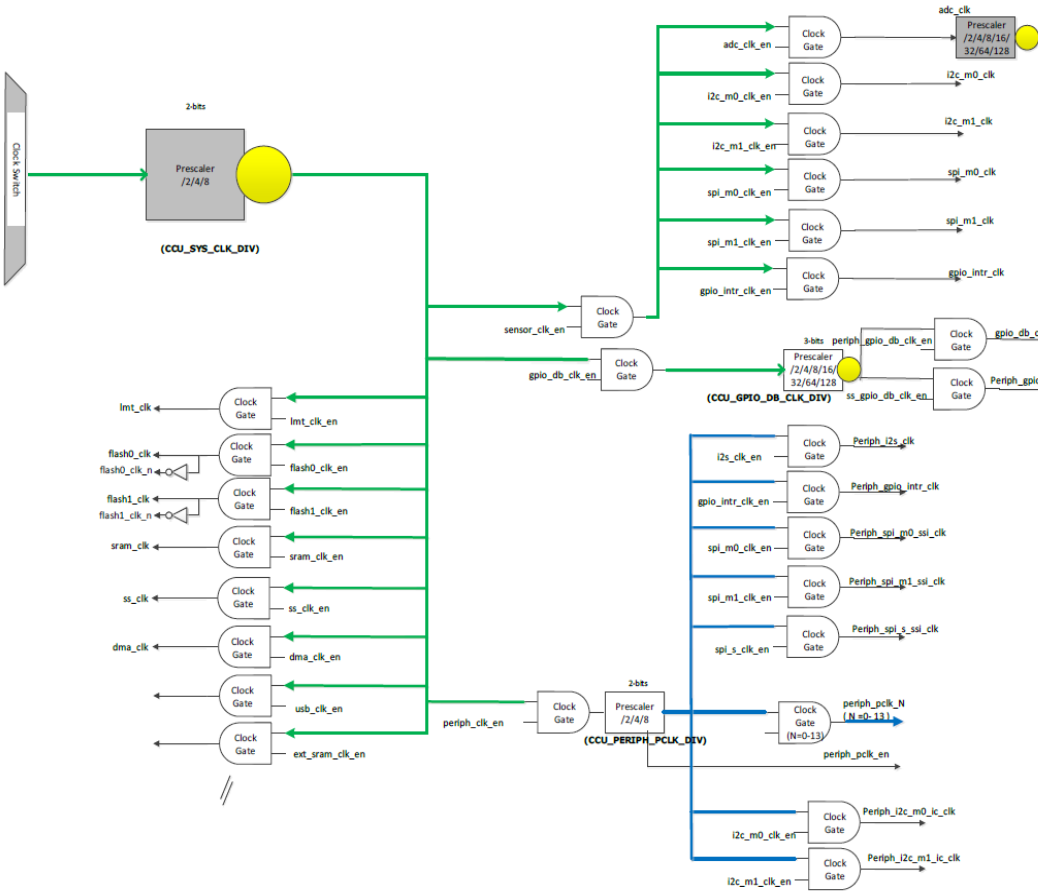


Figure 5.20 Intel Curie Peripheral's Clocking Diagram [127]

- **Alter the configured time delays**: Increasing the system clock speed barely affects some of the processes, such as the serial communication. This type of event is programmed with additional delays that ensure comfortable time limits within which the microcontroller operates flawlessly. To change these added delays, all source codes and device drivers related to the application were investigated and their respective default delays were reduced while carefully monitoring the overall performance.

- **Reconfigure the ADC block**: Added to the timing performances, the ADC's configuration plays an important role in having faster sampling rates. For example, lowering the ADC

resolution from 12 bits to 10 bits and lowering the reference voltage from 3.3V to 1.8V doubles the data acquisition speeds of the Intel Curie. However, such modifications lower the quality of the data collected. In this application, the ADC speed was improved by lowering the input capacitance of the signal conditioning module after adding the voltage-follower described in Figure 5.12. By doing so, the ADC block has a smaller RC time-constant ($\tau = RC$) that enables faster conversion times. Furthermore, all of the ADC's internal pull-up resistances were set active while their PWM capabilities were eliminated through the allocated control register 'IO_ADC0_CTRL' [127]. Finally, the ADC preprogrammed gain of 2 was eliminated by putting the entire A/D module in free-running mode to ensure a maximum bandwidth.

- **Tune the data transfer baud rate**: After increasing the ADC sampling speeds, it is necessary to increase the data transmission speeds between the microcontroller and the microprocessor. After all, the C1000 microcontroller has a limited built-in SRAM of 32 kB [127] which is not enough to hold sensory data of one engine cycle. To solve this problem, the serial communication speeds defined in the source codes were reconfigured, e.g. setting 'i2c_speed' variable to 'i2c_fast' in i2c.c source code. Then, all parity options that add overhead to the communication were eliminated. To further reduce the overhead, the readings coming from one analog (12-bit resolution) and one digital channel were combined together to form a 2-byte data packet. This maximizes the effective storage capacity of the buffer. Finally, the baud rate [146] of the UART controller was increased from its default value 9600 baud to 2M baud. Before making such change, it is important to note that baud rates tightly correlate with the base frequency of the system, 32 MHz, along with the defined pre-scaler divisors that lie between the UART peripheral and the system clock, as seen in Figure 5.20. In fact, this step was done carefully as the microcontroller can only accept a set of specific baud rates to ensure flawless data

communication. After increasing the baud rates, it is necessary to deal with another problem: buffer overflow. To go around it, the size of the communication buffer was increased dynamically from its original 16 Bytes to a length that maximizes the capabilities of the 32 kB SRAM when the code is running.

To sum up, the modifications carried above are the main drivers to increase the sampling rate of sensory system. Other adjustments include giving higher interrupt priorities to both sampling and communication events and copying all used source code functions to the main FDD firmware. This makes the microcontroller highly customized for the data-acquisition application.

After making all the aforementioned changes, the required sampling rate of 40 KHz was met. However, due to the intermittence in the data transfer, the recorded signals contained glitches. This was noticed after applying Fast Fourier Transform on the raw data. To eliminate the recorded discontinuity, the Intel Curie should sample and send the data simultaneously due to the limited RAM capabilities. However, this new scheme took around 45~47 microseconds which lead to setting the interrupts at 50-microsecond intervals (20 kHz sampling rate). Not discouraged, a new microcontroller was selected by giving much emphasis to its CPU maximum clocking speed as well as its RAM storage capabilities. The following section highlights this new implementation in details.

5.3.2. Second Attempt: The Teensy 3.6 Microcontroller DAQ Implementation

The Hardware Design

In this second attempt, the Teensy 3.6 development board was selected. Its CPU is the 32-bit, 180 MHz ARM Cortex-M4 processor. The board comes with a prepackaged bootloader that allows simple USB connection with the UDOO X86 board [147].

Figure 5.21 Teensy 3.6 Microcontroller [147]

The teensy 3.6 offers many features including a 1 Mbyte Flash memory to save the FDD firmware and a 256 Kbyte RAM memory to save intermediate buffer data for the serial communication. The main processor is the MK66FX1M0VMD18 chip which belongs to a high performance K66 sub-family arm processors [148]. This processor offers High USB communication speeds (480Mbit/sec) which match the speeds of the available USB 3.0 of the UDOO X86. The board also has 62 I/O pins amongst which there are 22 PWM outputs. When it comes to Teensy's ADC capabilities, the board has 2 ADC converters with 13-bit resolution and 2 DAC outputs with 12-bit resolution.

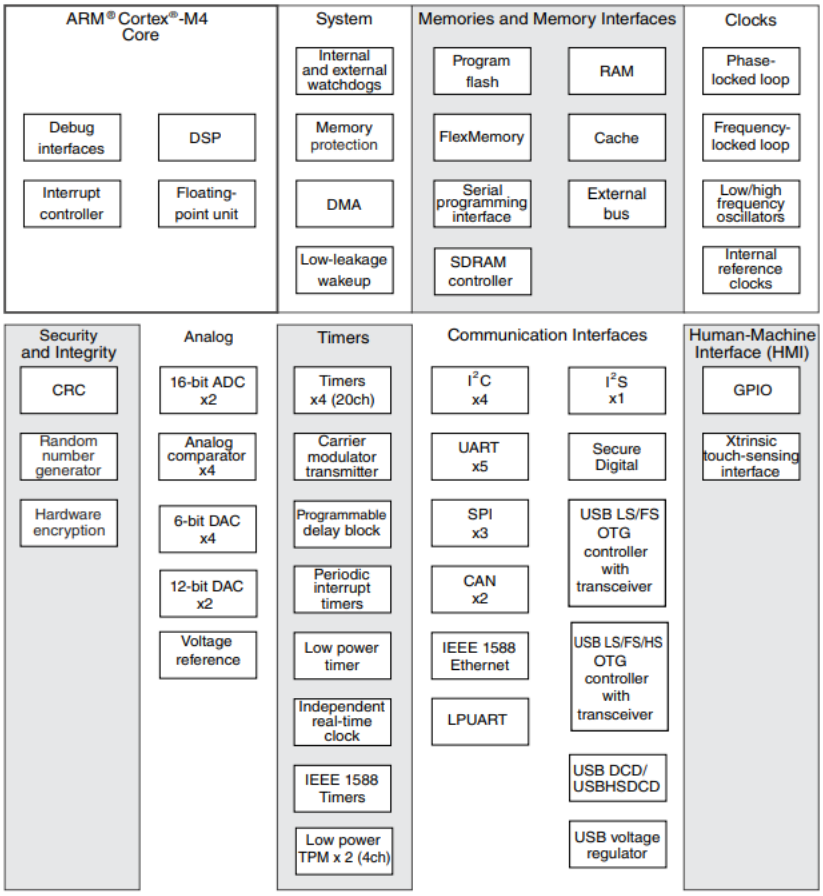


Figure 5.22 K66 Block Diagram [148]

Having 2 Analog-to-digital converters is not enough for FDD applications. The sensory system requires at least 8 analog channels (4 vibration + 4 sound). This limitation is circumvented using an external ADC module. The module uses the LTC 1867 chip from Analog Devices Inc. [149] that offers 8-channel of 16-bit A/D converters. All channels can be configured either for single-ended or for differential inputs. Added to that, the conversion can be either unipolar or bipolar to allow both positive and negative voltage readings. To use this module with the signal conditioning circuits designed before, the LTC 1867 channels were configured single-ended and unipolar. This module connects with the Teensy board using the SPI serial protocol that allows sample rates of up to 200 ksps (kilo-samples per second).

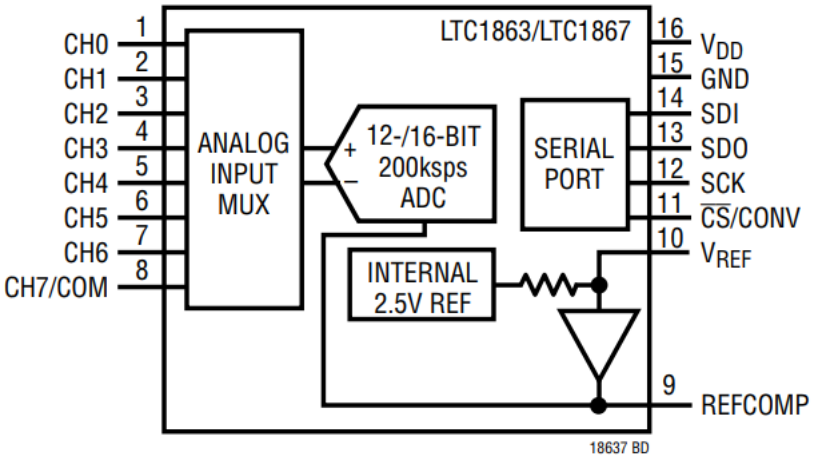


Figure 5.23 LTC 1867 Block Diagram [149]

With all the hardware components selected and their respective integration completed, the next step in the hardware design is to prepare a comprehensive circuit schematic that encompasses all these elements in one board. This board contains the signal conditioning for 4 accelerometers (Figure 5.12), the signal conditioning for both the crankshaft and the camshaft position sensors (Figure 5.14), and few circuit elements to provide the needed voltage rails for power purposes. The signal conditioning circuits are directly connected to the LTC 1867 ADC module. This module interfaces with the teensy 3.6 using SPI's 4 lines, MOSI, MISO, Clock, and Chip Select as shown in Figure 5.24.

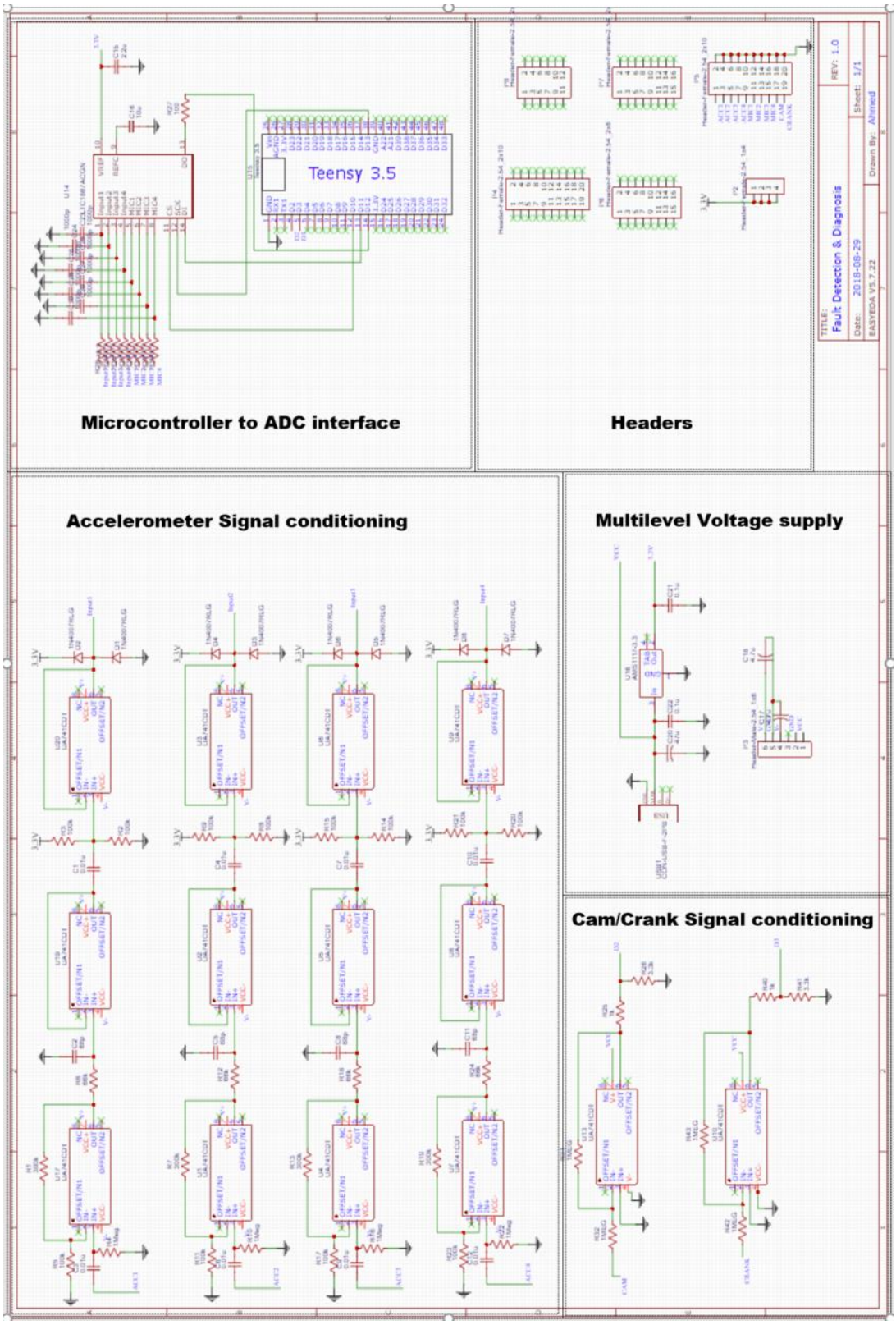


Figure 5.24 FDD System's Circuit Schematics

With the circuit schematics ready, all elements were placed into a double-layer general purpose PCB. To increase the SNR performances of the FDD system, both layers were fully grounded and both grounds were connected with the chassis of the vehicle. While the upper layer contains all signal conditioning for the accelerometers (Figure 5.12), the bottom layer contains the signal conditioning for the crank/cam (Figure 5.14) along with the ADC module, shown in Figure 5.25. The new PCB board is designed to host the teensy 3.6 on top of it, see Figure 5.26. The final prototype fits the UDOO as an add-on shield for its microcontroller side. While this solution offers an overall compact design, it remains practical as well by keeping the original GPIO pins of the UDOO exposed for other applications. The final UDOO X86 DAQ hardware is presented in Figure 5.27.

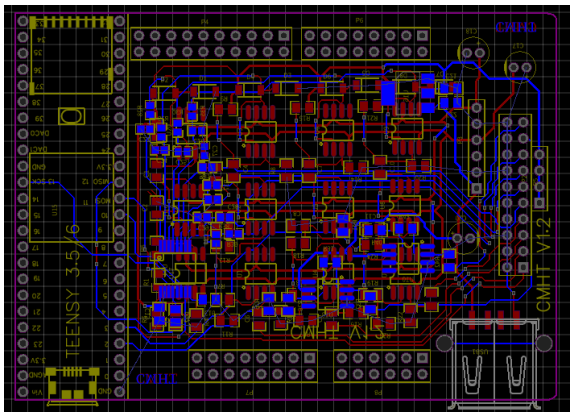


Figure 5.25 FDD System's PCB Wiring

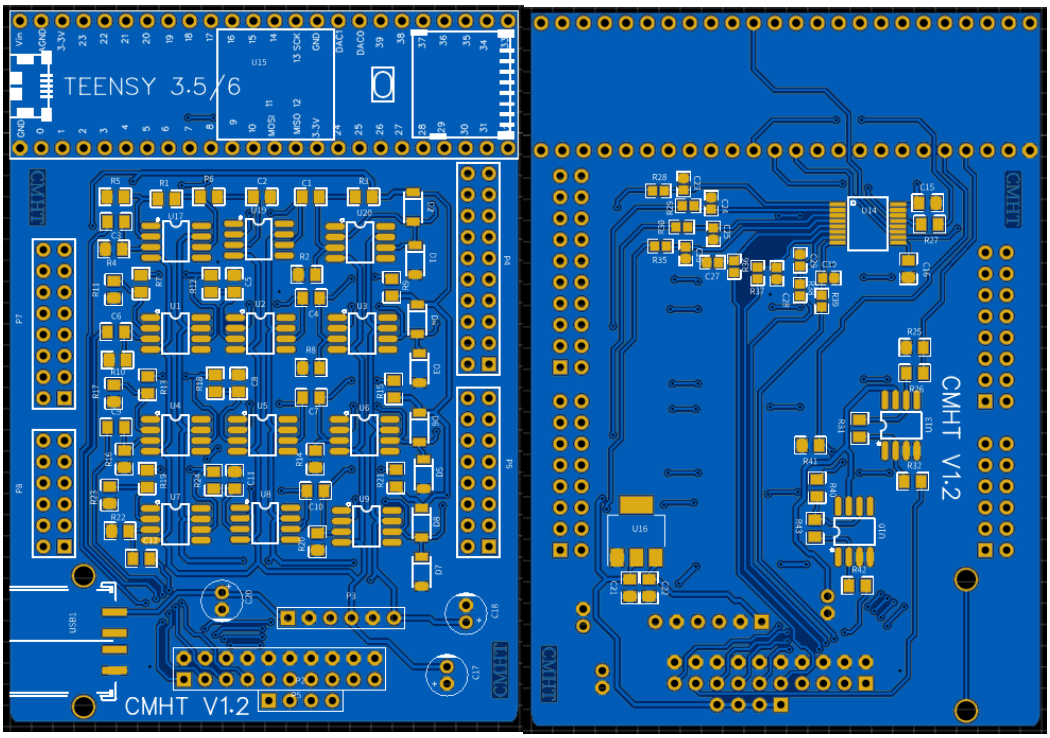


Figure 5.26 The FDD System Top and Bottom Layer

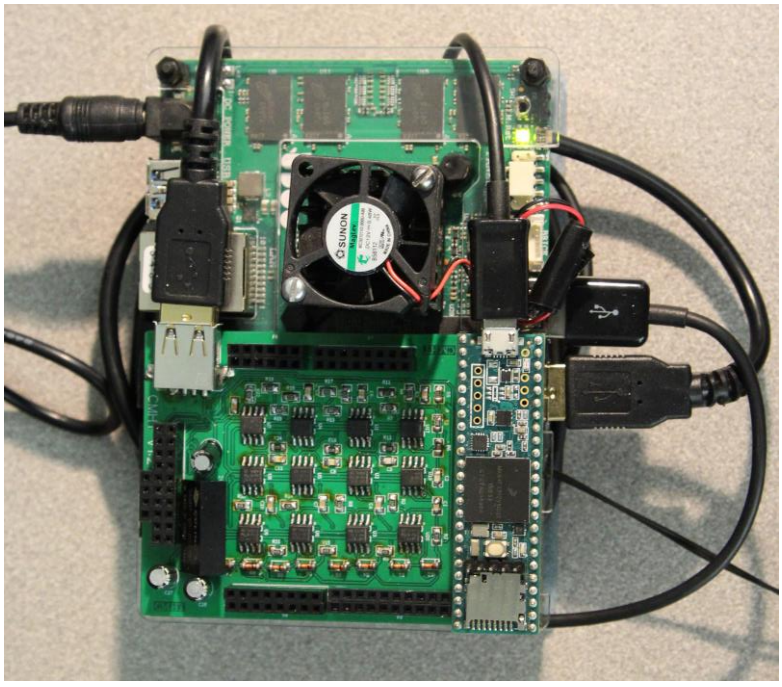


Figure 5.27 The FDD System DAQ Prototype

The Software Design

Similarly to the previous implementation, the software design relies on both firmware and application software. The choice of this second microcontroller provides two main advantages: the fast clocking speed that goes up to 180 MHz and the available 256 kB RAM. Faster clocking correlates with quicker execution times in all of A/D conversion and serial communication channels. On the other side, the increase in memory capacity allows to store up to 5 engine cycles of 6 different analog channel (16-bit each) at 20 kHz simultaneously. With these valuable updates in the hardware, the firmware for this set up offers to the user three main different sampling schemes to choose from. In all these options, the A/D resolution is set to 16 bits.

- 6 analog channels at 20 kHz simultaneously.
- 3 analog channels at 40 kHz (X2 using synchronization signals from Cam/Crankshaft position sensors).
- 1 analog channel at 125 kHz (X6 using synchronization signals from Cam/Crankshaft position sensors).

Similarly to the previous implementation, the FDD system samples the connected sensors using interrupts. This sampling scheme allows for a fixed sampling interval that is configurable by the user. Every interrupt callback function uses the SPI serial protocol to communicate with the external A/D module. This is a full duplex communication channel [150] where the microcontroller initiates the process by sending 2 bytes to the ADC for configuration (channel number, channel type: single-ended versus differential). On the receiving end, the ADC seizes message, configures itself accordingly, and sends the suitable ADC channel's output (16-bit resolution) back for storage. The microcontroller collects the data and store it in a FIFO buffer inside the RAM. Once the buffer is full, the microcontroller stops the interrupt routine and proceeds with emptying the buffer into

the USB channel that links it with the microprocessor. In case higher sampling rates are selected (40 and 125 kHz), the microcontroller repeats the above process until it logs all the attached sensors. The following flow diagram depicts in details one data acquisition cycle of all sensors.

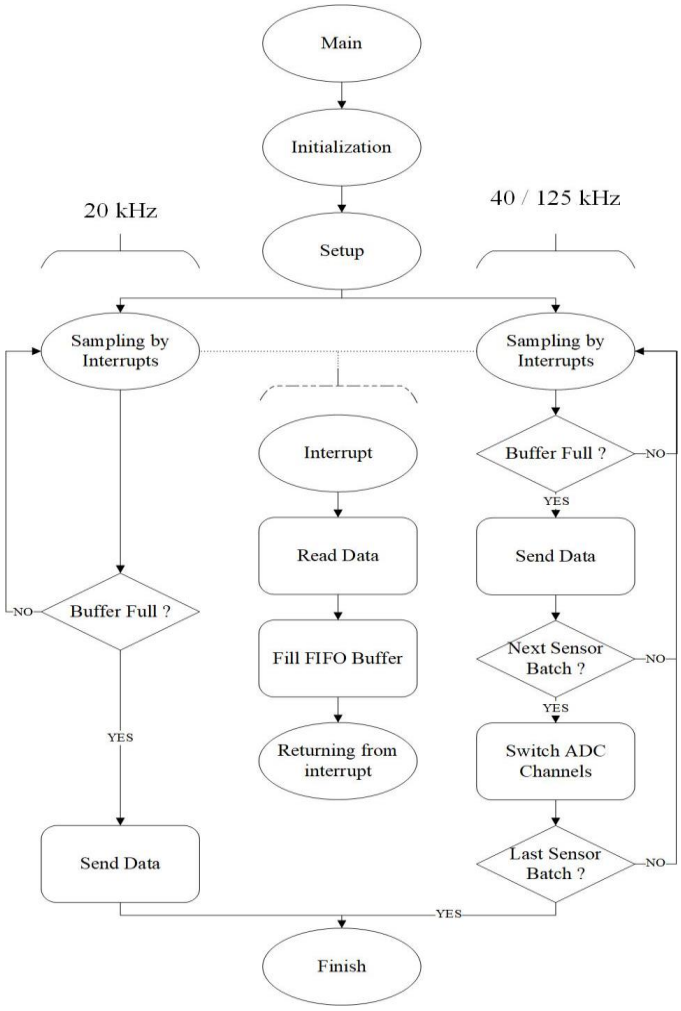


Figure 5.28 Teensy 3.6 Sampling Flow Chart

To reduce the amount of codes involved, the serial communication between the microcontroller and the microprocessor was first coded in MATLAB. This allowed having both the serial communication and the signal processing in one file. Unfortunately, after testing this implementation, it took on average 50 seconds to send a 2-second sampled data. This goes against

any practical real-time application of the DAQ system. So the second attempt was carried out using the C language. By applying the Termios serial library for POSIX operating systems [151], the data transmission overhead was reduced to 3 seconds. This additional firmware was compiled and loaded onto the Linux kernel with a high priority to tailor this general operating system to the DAQ application. Furthermore, to prevent the serial buffer from overflowing a full duplex communication between the microprocessor and the microcontroller was implemented. To this end, one cycle of data-collection averaged a 5-second period which allows the engine prognostics to be done in real-time. The following flow chart summarizes all elements in the software design of the FDD system.

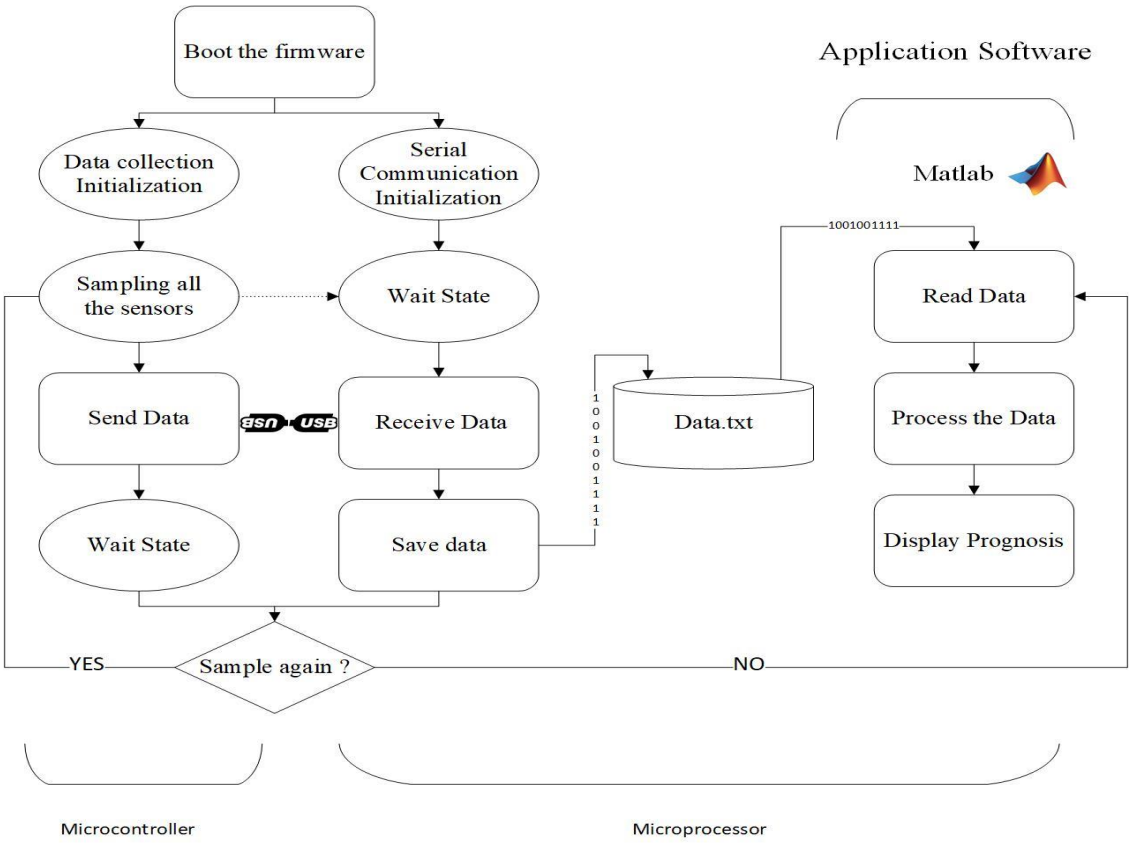


Figure 5.29 The FDD System Software Design Flow Diagram

When it comes to the FDD application software, all coding was carried in MATLAB. With that in mind, application software codes can be clustered into 2 categories: Offline Application Software and Online Application Software. The former category is responsible to generate all necessary elements required to carry the real-time online prognostics, e.g. train the neural network needed for the M1-Average-CAD FDD implementation or prepare a database of healthy-engine recordings for M2-CAD-IEMSPCA algorithm. Meanwhile, the latter category is responsible for carrying the diagnostics in real-time, and it provides the user with three main options to choose from: M2-CAD-IEMSPCA FDD which requires an extensive library of healthy recordings, M1-Average-CAD FDD which requires a pre-trained neural network model, and M3-CASA FDD which requires signals from both the cam and crankshaft position sensors. The last type of online application software is the least complex, the fastest, and yet the most accurate of the three in detecting misfire related faults. Furthermore, this algorithm bypasses the need to using a microprocessor and can be integrated within the firmware of the Teensy microcontroller directly. Finally, to enhance the usability of the DAQ system, the processed results are displayed for the user as follows.

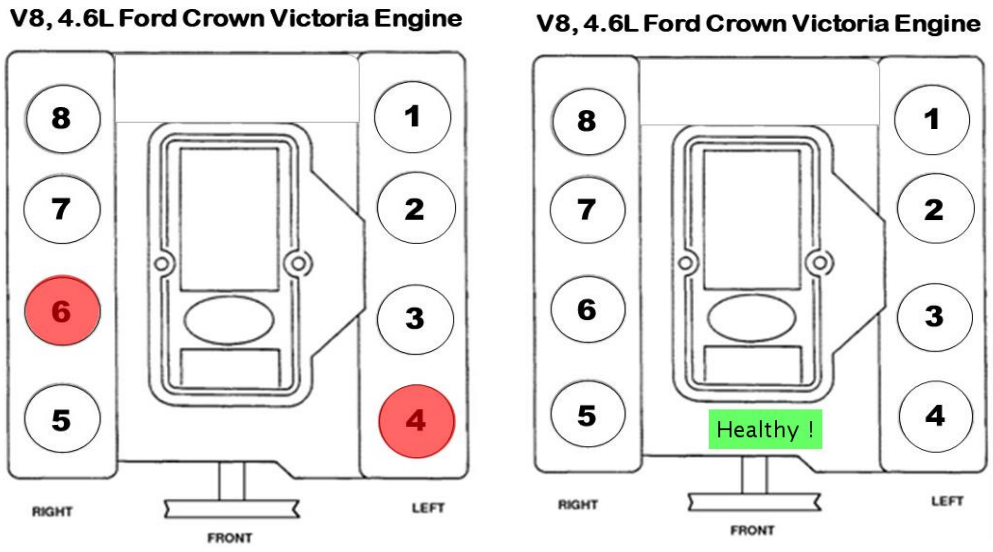


Figure 5.30 The FDD SystemUser Interface

5.4. Conclusion

This chapter starts by covering previous DAQ implementations within the Centre for Mechatronics and Hybrid Technologies at McMaster University for engine diagnostics. While the first attempt used a proprietary dSPACE hardware that was costly, the second implementation used a wireless microcontroller with limited computational capabilities and powering options. Both implementations were not practical as raw data needed to be communicated back to the user's computer for further processing. To have the processing done on board, a System on Chip (SoC) was selected, and it combines the computational power of both a microcontroller (for sampling) and a microprocessor (for processing). The selected board was then tailored for FDD. With this third attempt complete, the following table can summarize all Fault Detection and Diagnosis DAQ implementations.

	dSPACE	CC3200MOD	UDOO X86
Processor	µP: IBM PPC 750FX RISC (800 MHz)	µC: ARM Cortex-M4 (80MHz)	µP: Intel Pentium N3710 (2.56GHz) µC: ARM Cortex-M4 (180MHz)
ADC	16 Channels (12-bit, [0, 5V])	4 Channels (12-bit, [0,1.8V])	8 Channels (16-bit, [0, 3.3 V])
Sampling Rate	20 kHz Simultaneous	13 kHz Individual	20 kHz Simultaneous 40 kHz Simultaneous 125 kHz individual
Sampling Type	Sampling by Polling	Sampling by Interrupts	Sampling by Interrupts
Accelerometer	(4X) 1-axis, 34 kHz, ±50g, ±100mV/g	(1X) 3-axis, 6 kHz, ±25g, ±50mV/g	(4X) 1-axis, 34 kHz, ±50g, ±100mV/g
Microphone	No Microphone	(1X) 60 Hz to 15 kHz	(4X) 20 Hz to 20 kHz
Knock Sensor Interface	NO	NO	Yes
Synchronization	Camshaft	Camshaft	Cam + Crankshaft
Communication with user	Wired	Wireless	Wireless
Power supply	Mains power	Li-Po Battery power	Car Battery Power
Advantages	Powerful and robust	Compact and open source	Compact, open source, powerful
Disadvantages	Expensive, too many wires	Not powerful, Short life cycle	Not rugged, Relatively expensive

Table 5.2 Summary of All CMHT FDD Implementations

Chapter 6. Results and Discussions

This chapter starts by describing the experimental setups for Fault Detection and Diagnosis. Then, the results from the application of the three FDD strategies discussed in Chapter 3 are reported, namely for the M1-Average-CAD, the M2-CAD-IEMSPCA, and the M3-CASA algorithms. A comparison of their performance is then presented and discussed.

6.1. Experimental Setup

Two experimental setups were used in this research project:

- In-vehicle testing supported the main thrust of the experiments. The set up enabled FDD application to the operation of a 4.6L ICE within a Ford Crown Victoria vehicle. In this first setup, only single- and double-cylinder misfire conditions were physically simulated.

- The second experimental platform was a 5L Ford Coyote engine on an engine dynamometer at the Centre for Mechatronics and Hybrid Technology at McMaster University. This setup enabled data collection pertaining to advanced faults, such as knocking and pre-ignition, presenting challenging fault conditions for the FDD techniques.

In addition to the experimental platforms, this section covers the fault-simulation procedures. The tests involved physical simulation of faults on healthy engines to collect the data of multiple sensors over the course of at least one engine cycle.

6.1.1. Setup 1: The Ford Crown Victoria Engine

The first experiment setup hosts the FDD system, and it is summarized in Figure 5.16. The FDD data acquisition system is interfaced with three main elements: the engine, the car battery, and the user's computer. In its connection with the engine, The DAQ system samples all connected sensory

channels to perform the diagnostics. Both microphones and accelerometers are used externally to the engine and mounted accordingly on it. Meanwhile, all of camshaft, crankshaft, and knock sensors are proprietary to the ICE and require signal processing circuit to meet DAQ's electrical characteristics. To tap into these signals, the Ford Crown Victoria's electrical diagram was used. The power needs of the DAQ system are met by connecting it to the OBD-2 port. This connection incorporates an intermediate switching buck-boost configuration. Once the connection with the OBD-2 port is established, all voltages in both the sensory and the communication lines are referenced with the car's chassis ground. Finally, in this first setup, the DAQ system is controlled remotely through the user's computer.

The FDD system can connect a maximum of 4 accelerometers and 4 microphones with its ADC module. There are several possible positions and orientations for mounting these different sensors. In this regard, the microphones have flexible mounting options in comparison with the accelerometers as a direct contact with the engine block is not required. In this experiment, to exploit the inherent symmetry of the V8 engine, all sensory inputs were mounted on the right side of the engine block. Two of the accelerometers are mounted on the cylinder head, namely, the right front head (RFH), the right rear head (RRH) as shown by the blue coloring in Figure 6.1. These one-axis accelerometers were seated perpendicular to the piston movement. The microphones were placed to achieve a maximum coverage of the noise emitted by the engine's right bank. The setup used 3 microphones with different mounting locations, as depicted in Figure 6.2

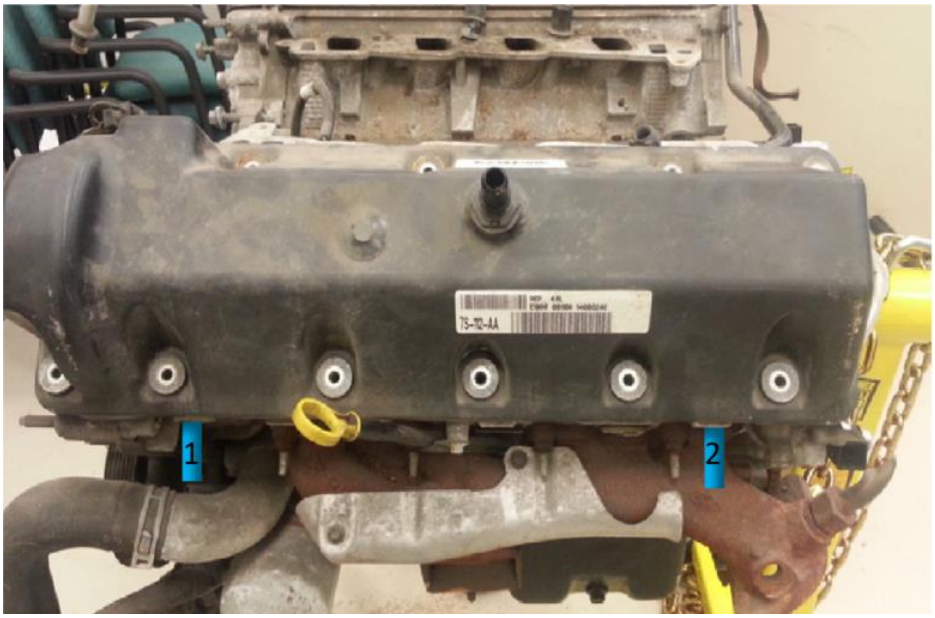


Figure 6.1 Mounting the Accelerometers [8]

Figure 6.2 Mounting the Microphones

After completing the instrumentation and hardware connections, the FDD sensory system and the accelerometers driver circuit are grouped inside one box that was placed in the front passenger seat as shown in Figure 6.3. Figure 6.4 illustrates a simplified diagram that shows the mounting location and direction of all *vibro-acoustic* sensors that were available in the first experimental setup.

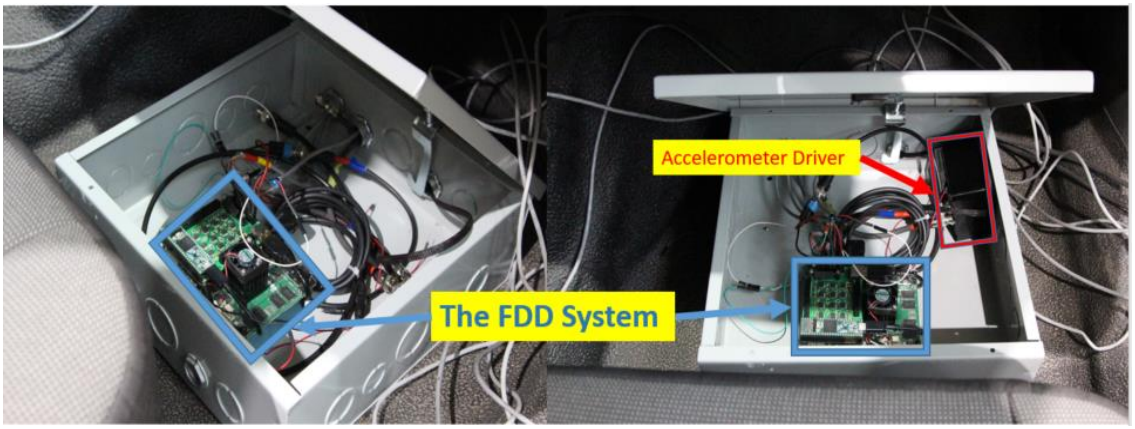


Figure 6.3 The FDD System Enclosure

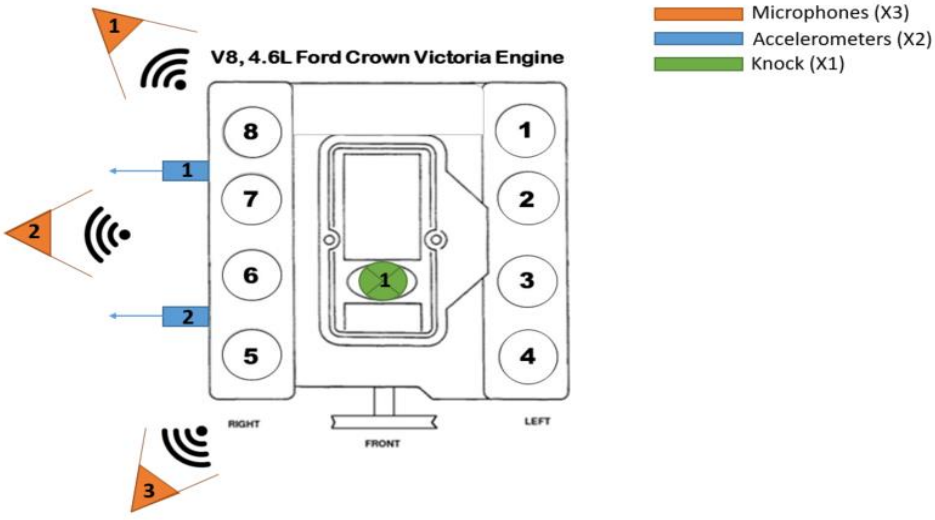


Figure 6.4 Simplified Sensor Diagram for Setup 1

6.1.2. Setup 2: The Ford Coyote Engine

This research project employed a secondary setup to further test the diagnostic capabilities of the FDD techniques. While the first setup was used to generate misfire faults, the second setup was used to introduce knocking and pre-ignition fault conditions. This secondary implementation is based on the Ford Coyote engine, with specifications summarized in Table 6.1. This engine is currently used in the Ford F150 truck and the Mustang sports car. The Ford Coyote engine was installed on the Titan T 250 dynamometer with specifications summarized by Table 6.2. The Kisler Kibox and its optical encoder were used to collect the data in crank-angle domain. Table 6.3 summarizes the environment in which the engine was placed.

Item	Specification
Displacement	5.038 L
No. of cylinders	8
Bore	93.0 mm (3.661 in)
Stroke	92.7 mm (3.649 in)
Firing order	1-3-7-2-6-5-4-8
Spark plug	12405
Spark plug gap	1.25-1.35 mm (0.049-0.053 in)
Compression ratio	12:1
Engine weight (W/O accessory drive components)	453.0 lb (205.5 kg)

Table 6.1 Ford Coyote Engine Specifications

Item	Specification
Torque	400 (600) Nm
Power	220 kW
Speed	8000 1/min
Idle speed	> 700 1/min
Mass moment of inertia	> 0.15 kgm2

Table 6.2 HORIBA Titan T 250 Engine Dynamometer

Facility	Explanation
Gasoline Engine	Gen 3, 2018 Ford Coyote Engine (5.0L 32V Ti-VCT/5.0L Ti-VCT V8 (308kW/418PS))
Engine Dynamometer	Horiba Schenk T 250
Emission Analyser	CAI
Engine Control Unit	EFI Technologies Open Access ECU
Combustion Analyzer	Kistler KiBox Combustion Analyzer
Crank Angle Position Sensor	Kistler Optical Encoder
Engine Dyno Accessories	Oil Cooling Circuit, Engine Coolant Circuit, Fuel Supply System, Electrical Power Supply System

Table 6.3 Second Experimental Setup Facility

In this second experimental setup, seven sensors were logged in order to conduct FDD testing. The engine has 4 internal knock sensors: Right Front Knock, Left Front Knock, Right Rear Knock, and Left Rear Knock. The mounting position and direction of these 4 sensors are shown in a simplified form in Figure 6.8. In addition to the knock sensors, two one-axis accelerometers and one microphone were added to engine. Figure 6.5 shows the overall setup, and it highlights the location of the microphone which is placed on top of the engine block pointing downwards as well as the location of the two accelerometers which were mounted on the right front side of the engine block (close to cylinder # 1). Figure 6.6 zooms on the mounting location and direction of the microphone. Furthermore, Figure 6.7 shows the 2 accelerometers which were attached to the engine through an intermediate bracket. This bracket allowed for two mounting options: accelerometer1 was mounted perpendicular to the piston movement, and accelerometer 2 was mounted in parallel with it. Having the accelerometers aligned in perpendicular directions provides useful insight fault detection and isolation. Figure 6.8 provides a simplified diagram of the 7 sensors used this secondary setup.

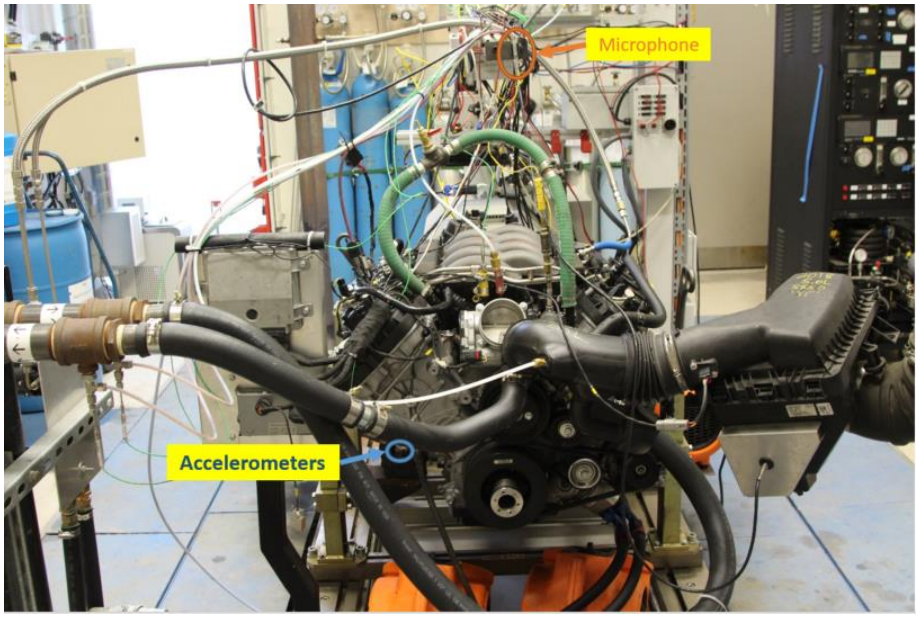


Figure 6.5 The Second Experimental Setup: Ford Coyote Engine

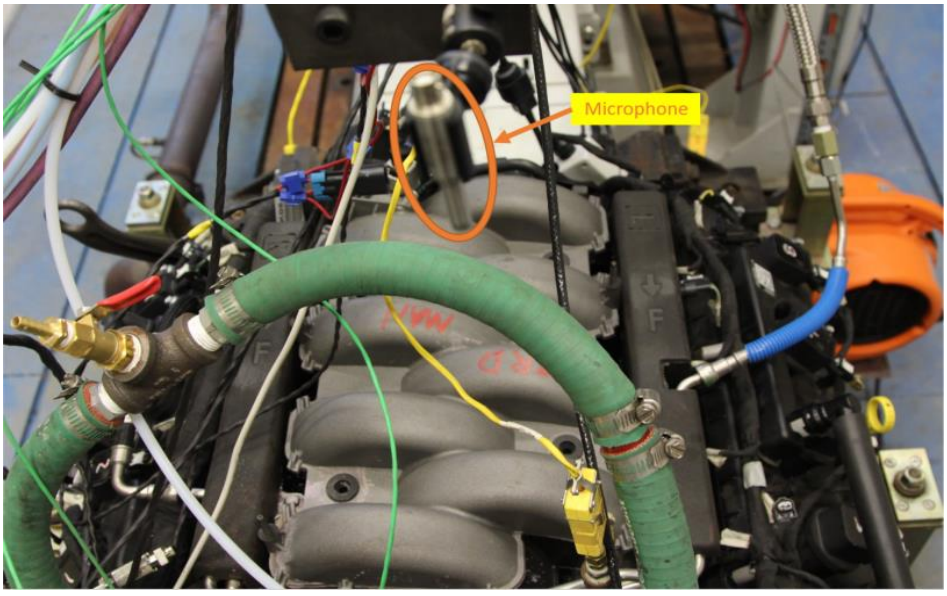


Figure 6.6 The Placement of the Microphone

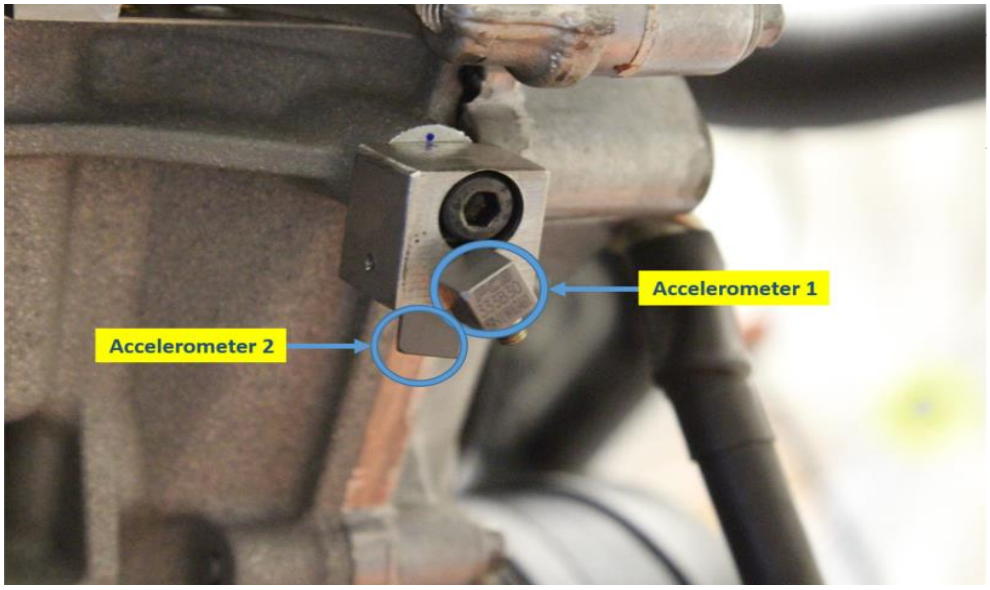


Figure 6.7 The Placement of the Accelerometers

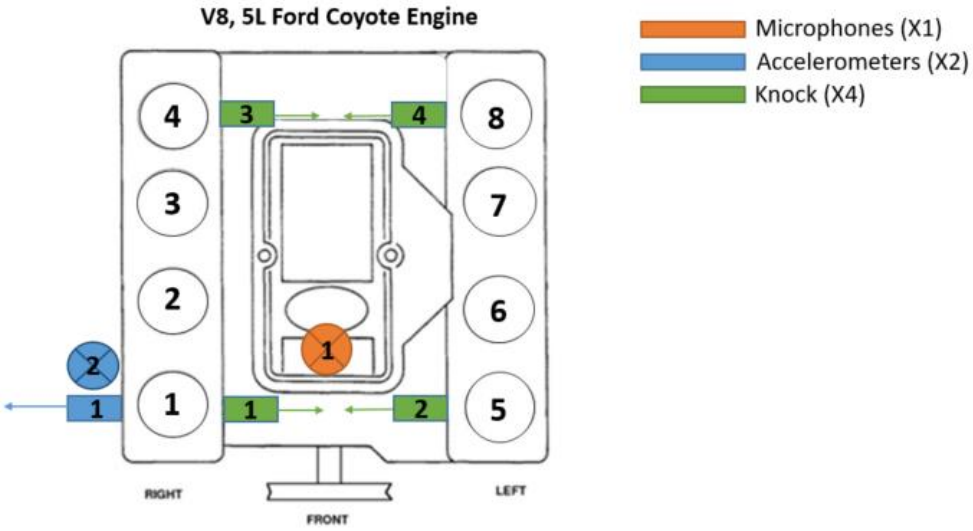


Figure 6.8 Simplified Sensor Diagram for Setup 2

The following Table 6.4 compares the two setups that were used to carry fault detection and diagnosis of internal combustion engines.

Specifications	Setup 1	Setup 2
Engine	Ford Crown Vic (V8, 4.6L)	Ford Coyote (V8, 5.038 L)
Engine Environment	Inside the Vehicle	On the Dynamometer
DAQ System	Udoo X86 Ultra	KiBox (from Kistler)
Sensors	Accelerometer (X2), Microphones (X3), Knock (X1), Crankshaft + Camshaft	Accelerometer (X2), Microphone (X1), Knock (X4), Crankshaft
Faults	Single and Double misfire faults	Pre-ignition and knocking faults
Fault simulation	Manually	Using Open Access ECU
Engine Speed	~800 rpm (idle)	3500 rpm

Table 6.4 Summary of FDD experimental Setups

6.1.3. ICE Fault Induction

To test the performance of the three FDD strategies considered in this research namely M1-Average-CAD, M2-CAD-IEMSPCA, and M3-CASA, different faults were physically simulated using the two experimental setups. There are mainly 3 tests that were carried:

- **Test 1**: 8 single-cylinder misfire conditions were simulated on the Ford Crown Victoria engine (setup 1).
- **Test 2**: 8 single- and 4 double-cylinder misfire conditions were introduced on the Ford Crown Victoria engine (setup 1).
- **Test 3**: knock and pre-ignition faults were simulated with different intensities on the Ford Coyote engine (setup 2).

For each test, every fault condition is labelled in order to train a neural network classifier. Table 6.5 summarizes the labelling scheme that was adopted. Furthermore, Figure 6.9 depicts the different cylinders' ordering schemes that were adopted for the two engines. In the case of the Ford Coyote

engine, cylinder 1 refers to the front cylinder which belongs to the right bank. Meanwhile, for the Ford Crown Victoria engine, the same labelling refers to the rear cylinder which sits at the left bank of the engine. Although the experiment was carried through three different tests, the M3-CASA technique was implemented with Test 1 and Test 2 only. As mentioned earlier, the Crank-Angle Speed Analysis technique was derived to detect misfire conditions only.

Condition Index	TEST 1	TEST 2	TEST 3
1	Cylinder 1 Misfire	Cylinder 1 Misfire	Light Knocking
2	Cylinder 2 Misfire	Cylinder 2 Misfire	Moderate Knocking
3	Cylinder 3 Misfire	Cylinder 3 Misfire	Large Knocking
4	Cylinder 4 Misfire	Cylinder 4 Misfire	Severe Knocking
5	Cylinder 5 Misfire	Cylinder 5 Misfire	Onset Knocking
6	Cylinder 6 Misfire	Cylinder 6 Misfire	Moderate Pre-ignition
7	Cylinder 7 Misfire	Cylinder 7 Misfire	Severe Pre-ignition
8	Cylinder 8 Misfire	Cylinder 8 Misfire	Healthy Condition
9	Healthy Condition	Cylinders 1 & 7 misfire	X
10	X	Cylinders 2 & 8 misfire	X
11	X	Cylinders 3 & 5 misfire	X
12	X	Cylinders 4 & 6 misfire	X
13	X	Healthy Condition	X

Table 6.5 Summary of the ICE Fault Conditions

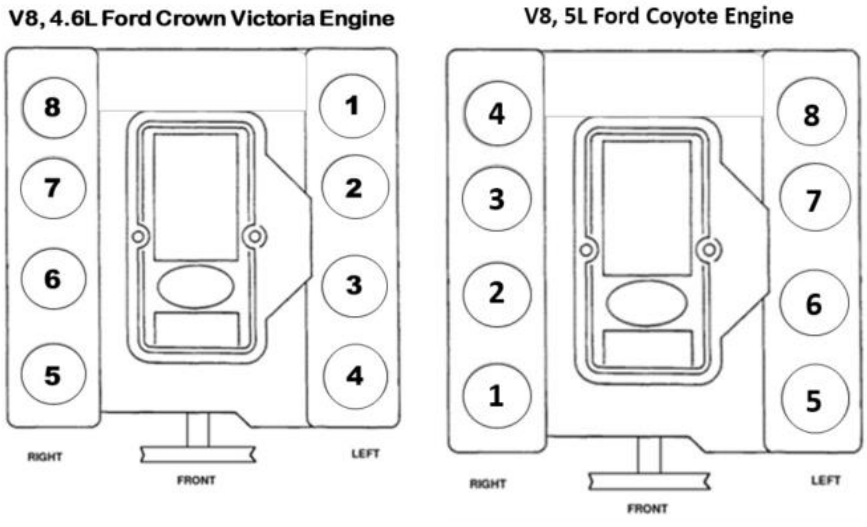


Figure 6.9 Ford Engines Cylinder Numbering Scheme [8]

	TEST 1	TEST 2	TEST 3
M1-Average-CAD	√	√	√
M2-CAD-IEMSPCA	√	√	√
M3-CASA	√	√	X

Table 6.6 Summary of the FDD Techniques used with the 3 Tests

To generate the faults summarized in Table 6.5, the two experimental setups followed a similar data collection procedure. Figure 6.10 illustrates the fault-simulation flowchart that was implemented as follows. Before acquiring the data, initially the engine needs to be in its healthy condition. While the engine is warming up to reach its normal operating conditions of oil and coolant temperatures, the data acquisition system is powered up, its communication with the user's control PC is established, and all its running firmware is compiled and booted. Once the engine reaches its normal operating condition, at least 50 engine cycles of healthy data are collected (setup1 at 800 rpm, setup2 at 3500 rpm). After collecting data on the healthy condition, the fault conditions are physically simulated on the engines. The experiment considers 3 different types of faults (misfire, knocking, and pre-ignition) and these are introduced to the engine as follows:

- To simulate the misfire fault in a particular cylinder chamber, the corresponding spark plug and oil injector are disconnected simultaneously in order to avoid unwanted accumulation of unburnt oil inside the combustion chamber.

- To simulate the knocking fault, the ignition timing of all cylinders are advanced using the Open Access ECU. This achieved knocking condition across all 8 cylinders of the Coyote engine.

- To simulate the pre-ignition fault, an inadequate spark plug (heat range) was installed in cylinder 1 of the Ford Coyote engine. As this modified component is unable to dissipate

heat effectively from one engine cycle to another, it ignites prior to the standard time of combustion.

After introducing faults in the system, data is collected and labelled according to the fault condition (min 50 engine cycles). This process was repeated until for all fault conditions are tested.

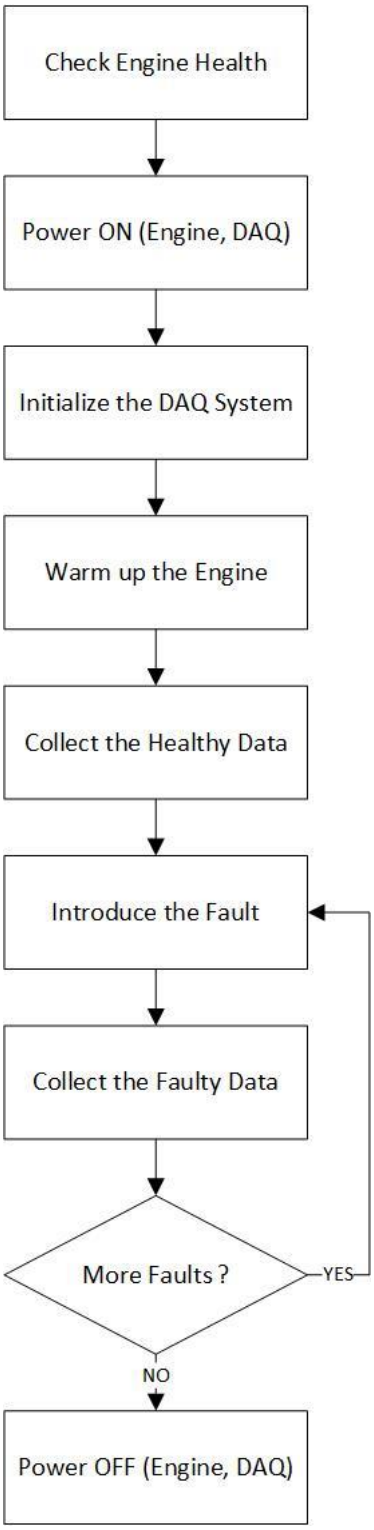


Figure 6.10 Data Collection Flow Chart

6.1.4. Fault Classification

Fault classification is carried by a Multi-Layer Perceptron neural network which is depicted by Figure 2.15. To prepare its training set, all the gathered data using the previous methodology (Figure 6.10) are labelled according to Table 6.5. The training is then carried out using the three methodologies discussed in Chapter 3, namely LM, SVSF, and EKF. In these experiments the structure of the neural network is different based on the test type (Test 1, 2, and 3) and the FDD technique (M1-Average-CAD and M2-CAD-IEMSPCA). The number of output nodes is dependent on the number of labelled categories as follows:

- Test1: the number of nodes in the output layer is equal to 9 (categories).
- Test 2: the number of nodes in the output layer is equal to 13 (categories).
- Test 3: the number of nodes in the output layer is equal to 8 (categories).

The number of nodes in the input layer is relative to the FDD techniques, and it is defined as follows:

- M1-Average-CAD: the number of nodes in the input layer is equal to 720 which represents one engine cycle's sample measurement (1° crank-angle resolution).
- M2-CAD-IEMSPCA: the number of nodes in the input layer is equal to the number of wavelet bins produced by the WPT ($2^{wavlet\ level}$) multiplied by the number of connected sensors to the FDD device. For example, the best performance was achieved using setup 2 which has 7 sensors when the wavelet level is set at 6. For that case, the number of nodes in the input layer is equal to 2^6 X 7 = 448 nodes.

As depicted by Figure 6.11, the neural network contains 3 layers (one input, one hidden, and one output layer). While the number of nodes in the input and output layers varies based on the test

type and the selected FDD technique, the number of nodes in the hidden layer is fixed to 10 nodes. The neural network is fully connected and its weights and biases initialized using Nguyen-window layer initialization function. This initialization scheme is done in Matlab using the 'initnw' parameter option. While the activation function for the output layer is pure linear transfer function, the activation function used with the remaining layers (input and hidden layers) is the log-sigmoid transfer function described by Figure 2.14.

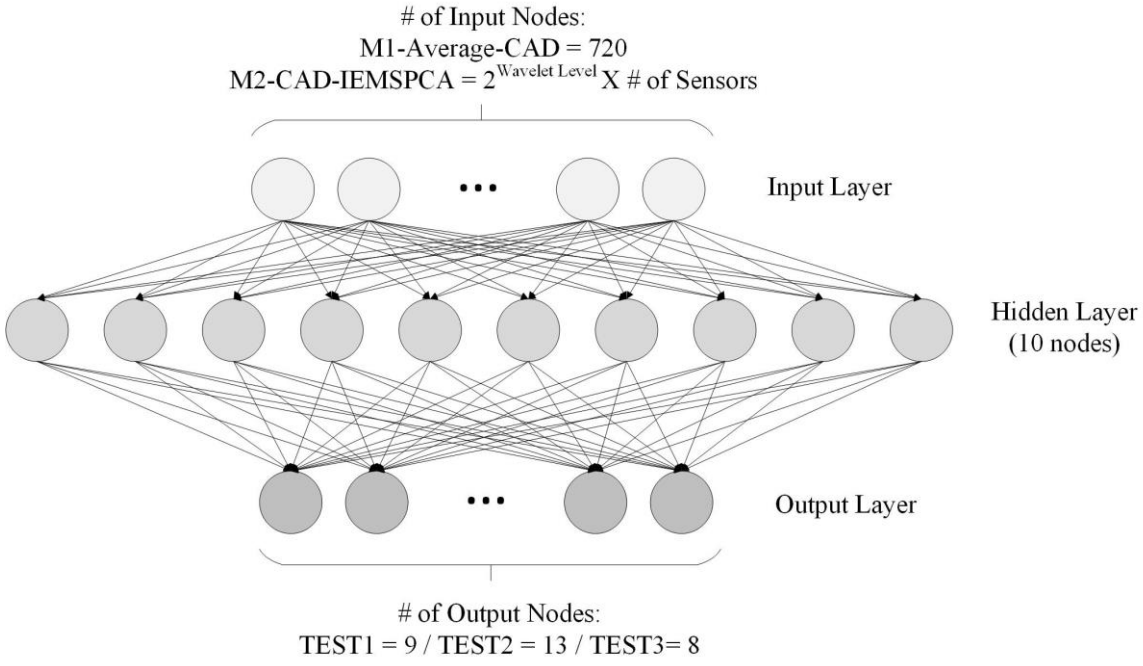
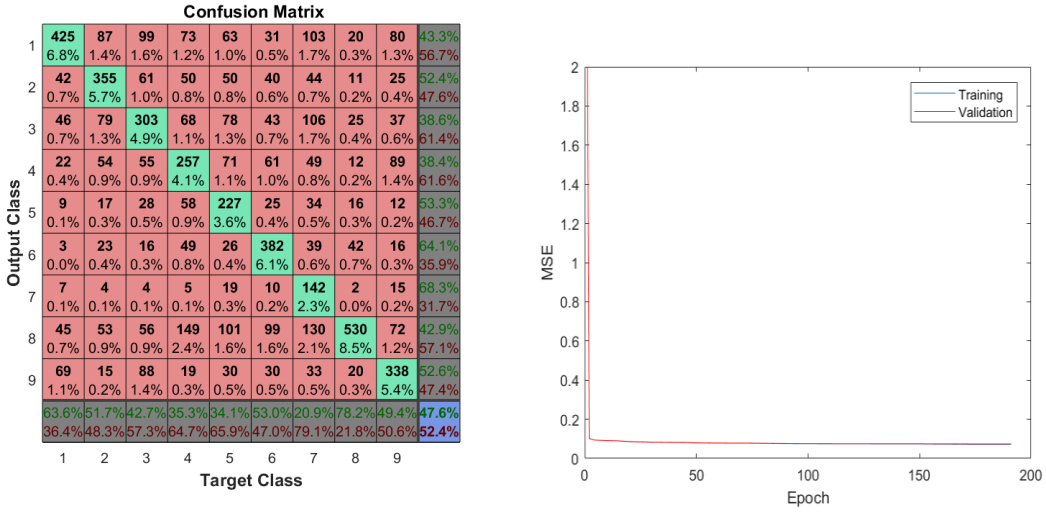


Figure 6.11 Multi-Layer Perceptron Structure

To assess the diagnostic capabilities of the above network, two main charts are compiled: the Confusion Matrix and the Mean Square Error (MSE) graph. Confusion matrices are validation tools of the network's training error with respect to a set of labelled data. This tool isolates the contribution of every error that happens during the classification task. These errors directly reflect the similarity between the different simulated conditions and isolate the type of faults that the

network is unable to discern. On the other hand, MSE graphs highlight the number of epochs needed to fully train the network. The training is finished when one of the training criteria is met. These criteria are, the maximum number of iterations (epochs), the maximum training time, the minimum gradient error, the minimum mean square error, and the maximum number of validation checks. Aside from the classification accuracy and the number of training epoch, other performance criteria include the duration of the training and the number of correct test cases.



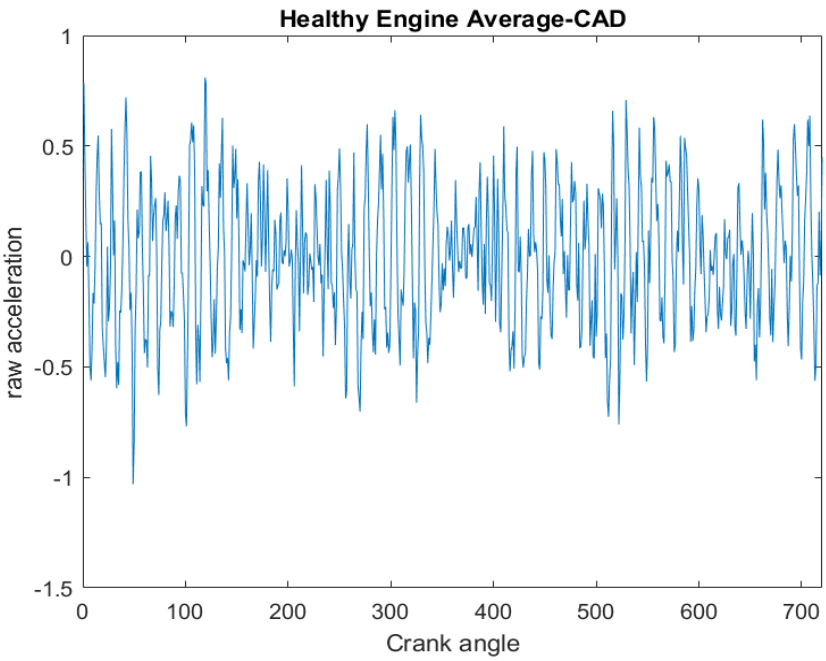
(a) ***(b)***

Figure 6.12 Validation Metrics: (a) Confusion Matrix, (b) MSE Graph

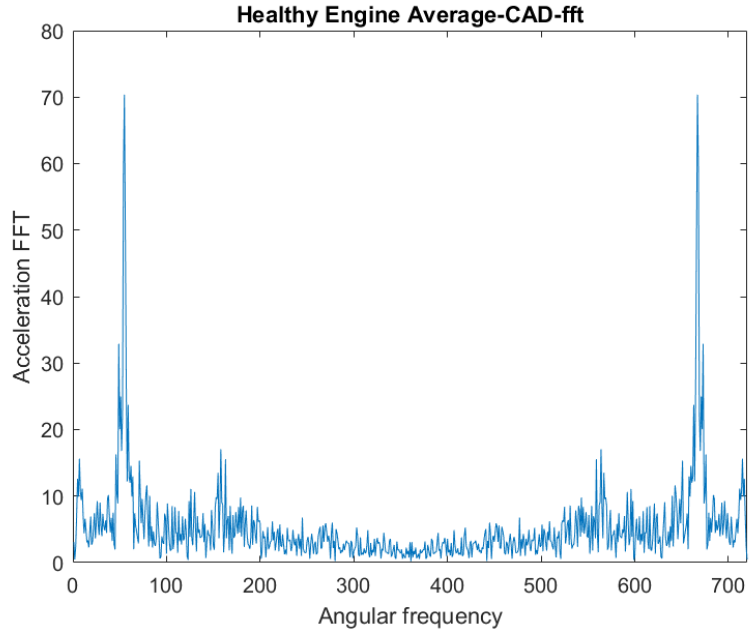
6.2. M1-Average-CAD FDD Results

The M1-Average-CAD FDD technique is used to carry the diagnostics of the engine using sensors separately. This technique highlights the contribution of each sensor using time, frequency, and time-frequency generated features. These features are all derived from the CAD transform of the vibro-acoustic raw data. This first FDD methods uses three different feature extraction components: Average-CAD (time), Average-CAD-FFT (frequency), and Average-CAD-WT (time-frequency).

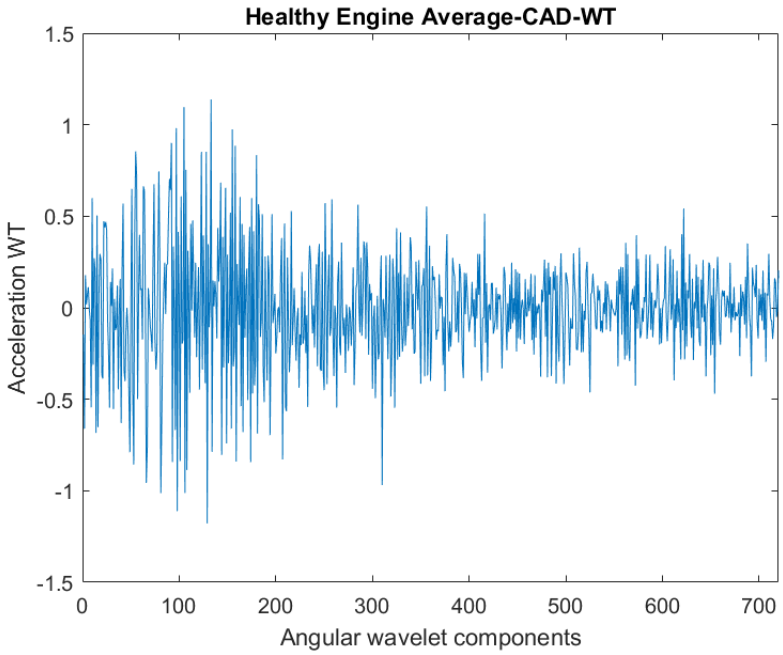
The following Figure 6.13 plots an example of the generated features from the accelerometer when the engine is healthy. The first subplot (a) depicts the CAD transform of the vibration data that was sampled at 200 kHz. The CAD transform interpolates the raw vibration (time) into 720 angular samples which make up one engine cycle at 1° resolution of crank angle. The following subplots, namely (b) Average-CAD-FFT and (c) Average-CAD-WT, are respectively generated by applying the Fast Fourier Transform and the Wavelet Transform on the Average-CAD data.



(a) Average-CAD



(b) Average-CAD-FFT



(c) Average-CAD-WT

Figure 6.13 The Three Features of M1-Average-CAD FDD Technique

The above features are generated for all fault conditions and are used separately to train The Multi-Layer Perceptron classifier. This FDD technique is implemented with the three test scenarios which are summarized in Table 6.5.

Test 1: Single Misfire Condition

The first test was carried in setup 1 where 8 single cylinder misfire conditions were introduced on the Ford Crown Victoria engine. To classify the extracted features using M1-Average-CAD FDD technique, three training methods were chosen: the LM, the EKF, and the SVSF algorithms. The implementation of these training algorithms were derived from the MATLAB codes which were developed by Feng [8]. Table 6.7 summarizes all combinations of training algorithms and feature extraction FDD techniques that were implemented in this first test. As mentioned in section 3.3, M1-Average-CAD FDD technique is carried out for each recording (sensor) separately. For the same sensory data, 3 training algorithms were implemented. Each training algorithm used either of Average-CAD, Average-CAD-FFT, or Average-CAD-WT generated features.

Sensors	Training Algorithms	Feature Extraction
Knock (X1) Accelerometer (X2) Microphone (X3)	LM	Average-CAD
		Average-CAD-FFT
		Average-CAD-WT
	EKF	Average-CAD
		Average-CAD-FFT
		Average-CAD-WT
	SVSF	Average-CAD
		Average-CAD-FFT
		Average-CAD-WT

Table 6.7 Test 1 Framework

Table 6.8 reports the training parameters used with the MLP Neural Network. In this test, 50 engine cycles were collected for each fault condition. Therefore, a total of 450 training features (50 engine cycle X 9 categories) were generated. These features constitute the training dataset for the MLP, and they are separated into three sets: the first set (270 features [60%]) is used to train the network, the second set (90 features [20%]) is used to validate (verify) the training of the network, and the third set (90 features [20%]) is solely used for testing purposes. That is to say, once the training is complete, the testing data is fed to the network to assess its classification performance. Other parameters include the SVSF's boundary thickness and conversion rate as well as the number of hidden layers that constitute the MLP network. Additionally, Table 6.8 includes 3 parameters that are used to stop the training of the network, and they are: the maximum number of training epoch which was set to 20, the minimum RMS error threshold which was set to 10^{-3}, and the minimum gradient error threshold which was set to 10^{-5}.

Parameter	Value
Total Number of Features	450
Ratio of Training : Validation : Testing	60% : 20% : 20%
Maximum Number of Epochs	20
Stopping RMS Error	10^{-3}
Stopping Gradient Error	10^{-5}
SVSF Conversion Rate	0.1
SVSF Boundary Thickness	0.1

Table 6.8 Test 1 Parameters

Test 1 results are summarized in Table 6.9 as follows:

Sensor	Training Algorithm	Feature Extraction	Accuracy	Training Epochs	Training Time (sec)	# of Failed tests /90
Knock	LM	AVG-CAD	100%	12	275	0
		AVG-CAD-FFT	100%	11	339	1
		AVG-CAD-WT	100%	12	345	0
	SVSF	AVG-CAD	11.1%	3	415	80
		AVG-CAD-FFT	11.1%	3	417	80
		AVG-CAD-WT	0%	3	429	90
	EKF	AVG-CAD	11.1%	2	211	79
		AVG-CAD-FFT	13.3%	2	211	79
		AVG-CAD-WT	11.1%	2	212	80
Accelerometer 1	LM	AVG-CAD	100%	9	207	0
		AVG-CAD-FFT	100%	10	235	0
		AVG-CAD-WT	100%	12	338	0
	SVSF	AVG-CAD	11.1%	3	416	80
		AVG-CAD-FFT	16.7%	3	416	74
		AVG-CAD-WT	11.1%	3	418	80
	EKF	AVG-CAD	11.1%	2	211	80
		AVG-CAD-FFT	12.2%	2	211	79
		AVG-CAD-WT	5.6%	2	212	84
Accelerometer 2	LM	AVG-CAD	100%	14	365	0
		AVG-CAD-FFT	100%	12	281	1
		AVG-CAD-WT	100%	10	308	0
	SVSF	AVG-CAD	11.1%	3	415	80
		AVG-CAD-FFT	10%	3	417	79
		AVG-CAD-WT	11.1%	3	417	80
	EKF	AVG-CAD	11.1%	2	211	80
		AVG-CAD-FFT	15.6%	2	213	75
		AVG-CAD-WT	11.1%	2	212	80
Microphone 1	LM	AVG-CAD	90%	20	682	6
		AVG-CAD-FFT	100%	14	330	1
		AVG-CAD-WT	100%	21	700	0
	SVSF	AVG-CAD	15.6%	3	417	79
		AVG-CAD-FFT	5.6%	3	416	85
		AVG-CAD-WT	11.1%	3	415	80
	EKF	AVG-CAD	11.1%	2	212	80
		AVG-CAD-FFT	11.1%	2	212	80
		AVG-CAD-WT	11.1%	2	212	80
Microphone 2	LM	AVG-CAD	100%	6	158	0
		AVG-CAD-FFT	100%	14	340	0
		AVG-CAD-WT	100%	5	131	0
	SVSF	AVG-CAD	15.6%	3	415	75
		AVG-CAD-FFT	17.8%	3	417	75
		AVG-CAD-WT	11.1%	3	417	80
	EKF	AVG-CAD	11.1%	2	211	80

		AVG-CAD-FFT	10%	2	212	81
		AVG-CAD-WT	12.2%	2	213	80
Microphone 3	LM	AVG-CAD	100%	8	247	0
		AVG-CAD-FFT	97.2%	10	240	0
		AVG-CAD-WT	100%	9	259	0
	SVSF	AVG-CAD	11.1%	3	531	80
		AVG-CAD-FFT	11.1%	3	447	80
		AVG-CAD-WT	11.1%	3	498	80
	EKF	AVG-CAD	13.3%	2	216	81
		AVG-CAD-FFT	20%	2	215	72
		AVG-CAD-WT	11.1%	2	217	80

Table 6.9 Test 1 Results

The following figures present the confusion matrices and MSE graphs for Acceleromter1 using Average-CAD feature components:

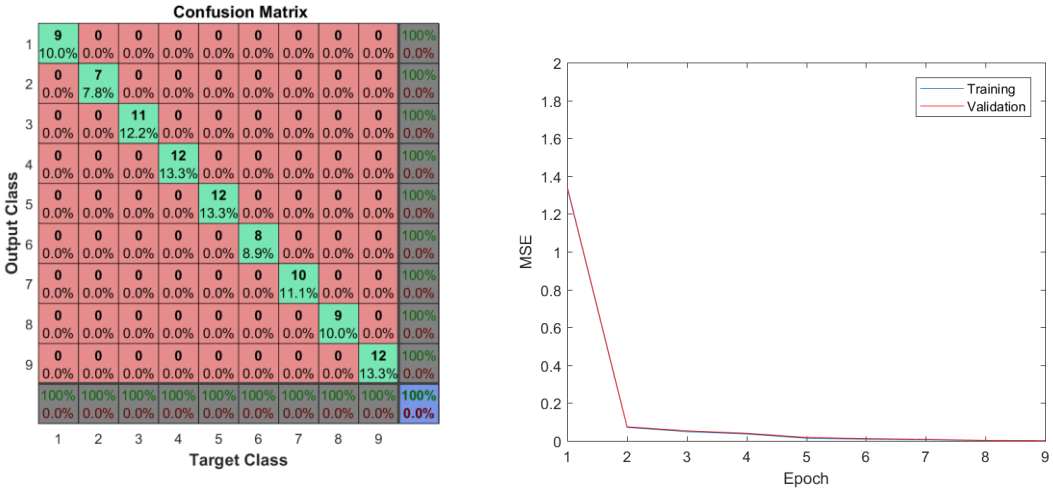


Figure 6.14 Accelerometer 1 Average-CAD LM Training Results

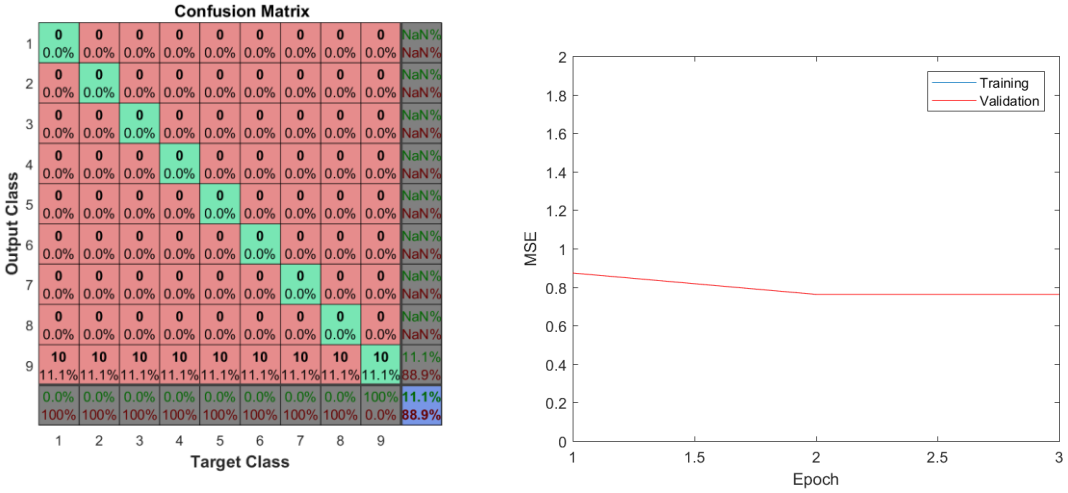


Figure 6.15 Accelerometer 1 Average-CAD SVSF Training Results

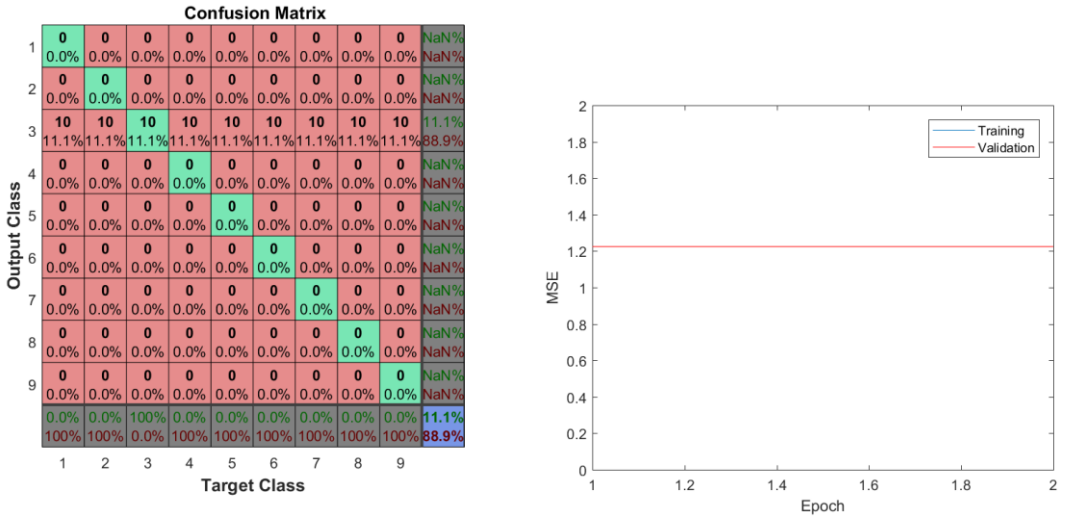


Figure 6.16 Accelerometer 1 Average-CAD EKF Training Results

Taking into consideration the above results, the following observations are made:

- The LM algorithm achieved the best results among the 3 considered training algorithms

- On a consistency basis, it took the SVSF 3 epochs and the EKF 2 epochs to reach the stopping condition of minimum gradient error without fully training the network.

- By isolating the number of training epochs and the total training time, it is clear that both EKF and SVSF are slower than LM algorithm.

- From the LM results, it can be deduced that microphone 1, when compared with the remaining 5 sensors, had the worst performance. This could be related to its location in the car which is highlighted by Figure 6.2. Nevertheless, all sensors classified the 90 tests cases with high accuracy (error < 6.66%).

- The two accelerometers and the knock sensor achieved similar classification performances which affirms the ability of the knock sensor to carry misfire FDD.

- All of Average-CAD, Average-CAD-FFT, and Average-CAD-WT feature extraction algorithms achieved similar performances. However, generating the features using the Average-CAD technique is simpler than the other two which require an additional step to compute the Fourier transform (Average-CAD-FFT) and the Wavelet transform (Average-CAD-WT).

Test 1 demonstrates that all *vibro-acoustic* sensors were able to individually achieve high classification accuracies of the 9 conditions (8 misfire + 1 healthy). The knock sensor, which is usually used to detect knocking conditions in the vehicle, can also detect the occurrence of all single-cylinder misfire faults. What is more, this first test showed that only LM algorithm was able to achieve acceptable training characteristics (speed and accuracy) using the labelled data. For that reason, the subsequent tests apply the Levenberg–Marquardt algorithm only to carry the classification of the neural network.

Test 2: Double Misfire Fault Condition

In the second test, 8 single- and 4 double- cylinder misfire faults were simulated on the Ford Crown Victoria engine (setup1). With the addition of the healthy condition, the neural network should classify 13 different categories. From Table 6.5, it is obvious that test 2 presents a challenging classification task for the network; all double misfire faults are relatively similar to single cylinder misfire conditions. For example, fault category 10 (dual misfire 2 and 8) is made up of fault categories 2 (single misfire 2) and 8 (single misfire 8). To account for the increased total number of categories (test 1= 9, test 2= 13), the number of training epochs was increased from 20 to 30. The remaining parameters are similar to those presented in test 1 which allows to compare the results from the two tests later on.

Sensors	Training Algorithm	Feature Extraction
Knock (X1) Accelerometer (X2) Microphone (X2)	LM	Average-CAD
		Average-CAD-FFT
		Average-CAD-WT

Table 6.10 Test 2 Framework

Parameter	Value
Total Number of Features	635
Ratio of Training : Validation : Testing	60% : 20% : 20%
Maximum Number of Epochs	30
Stopping RMS Error	10^{-3}
Stopping Gradient Error	10^{-5}

Table 6.11 Test 2 Parameters

Sensors	Feature Extraction	Accuracy	Training epochs	Training Time (sec)	# of Failed Tests /127
Knock	AVG-CAD	98.4%	20	698	2
	AVG-CAD-FFT	70.9%	30	1115	42
	AVG-CAD-WT	88.2%	18	1096	20
Accelerometer 1	AVG-CAD	100%	19	604	0
	AVG-CAD-FFT	89%	30	1485	15

	AVG-CAD-WT	90.6%	30	628	0
Accelerometer 2	AVG-CAD	95.3%	18	567	7
	AVG-CAD-FFT	50.4%	30	1346	65
	AVG-CAD-WT	90.6%	19	944	14
Microphone 1	AVG-CAD	97.1%	20	708	3
	AVG-CAD-FFT	76.4%	27	946	28
	AVG-CAD-WT	95.3%	22	762	14
Microphone 2	AVG-CAD	96.1%	14	467	2
	AVG-CAD-FFT	93.7%	19	665	13
	AVG-CAD-WT	88.2%	17	570	18

Table 6.12 Test 2 Results

The following figures are samples of the obtained confusion matrices and MSE graphs for Knock, Accelerometer 1, and Accelerometer 2:

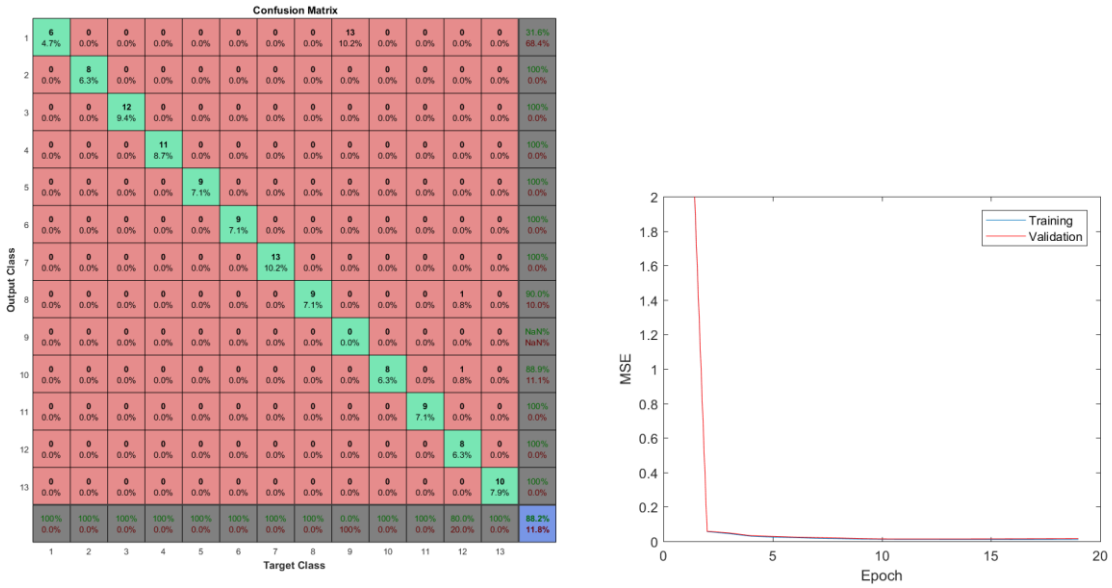


Figure 6.17 Knock Average-CAD-WT LM Training Results

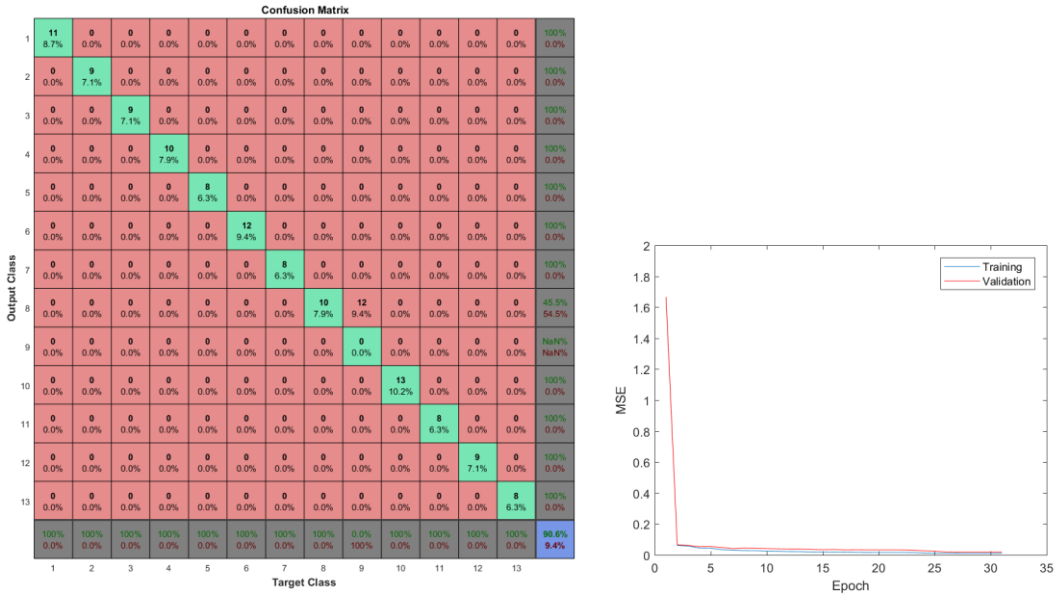


Figure 6.18 Accelerometer 1 Average-CAD-WT LM Training Results

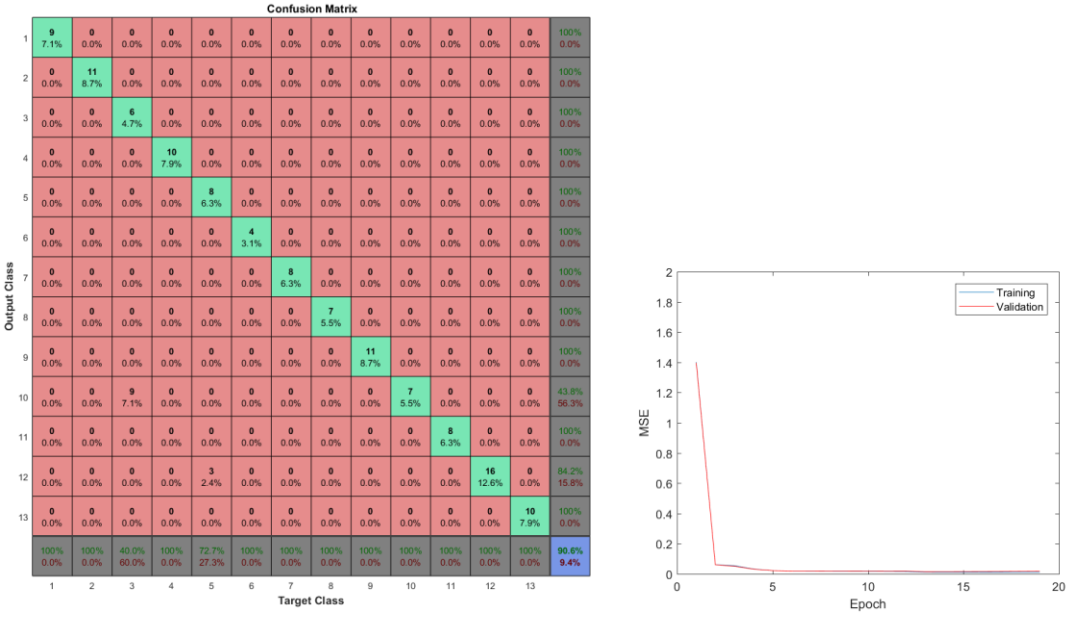


Figure 6.19 Accelerometer 2 Average-CAD-WT LM Training Results

Taking into consideration test 1 and test 2 results, the following observations are drawn:

- The results obtained in test 1 are considerably better than those of test 2. In fact, in this second test, only accelerometer 1 was able to achieve an accuracy of 100%. Furthermore, both the number of training epochs and training time increased considerably for all sensors.

- The highest number of failed tests was recorded by accelerometer 2 which had a success rate of 48.8%.

- Except for microphone 2, Average-CAD-FFT technique had the worst performance among the three feature extraction methods.

- Average-CAD consistently produced the best classification results across all sensors with the lowest accuracy level of 95.3% (accelerometer 2).

- The knock sensor achieved better classification performances than accelerometer 2.

- Similar to test 1, microphone1 ability to classify misfire faults was lower than that of microphone 2

In this second test, M1-Average-CAD FDD technique detected and classified single and dual misfire conditions simultaneously. Only Accelerometer 1 was able to achieve 100% accuracy which correlates with its location on the engine block. Similarly, microphone 1's mounting position affected considerably its ability to do FDD. Finally, processing the Average-CAD features further using FFT or WT did not achieve better FDD diagnostics of misfire conditions.

Test 3: Knock & Pre-ignition Fault Condition

In the third test, knock and pre-ignition faults were simulated at different intensities on the Ford Coyote Engine (setup 2). According to Table 6.5, the knocking fault occurred across all engine cylinders and was introduced at 5 levels of severity (category 1 ~ 5). On the other hand, the pre-

ignition fault was applied to cylinder #1 with 2 levels of intensity (category 6 ~ 7). As the number of faults is relatively similar between Test 1 and 3, their parameters are identically set as shown in Table 6.14.

Sensors	Training Algorithm	Feature Extraction
Knock (X4) Microphone (X1) Accelerometer (X2)	LM	Average-CAD
		Average-CAD-FFT
		Average-CAD-WT

Table 6.13 Test 3 Framework

Parameter	Value
Total Number of Features	720
Ratio of Training : Validation : Testing	60% : 20% : 20%
Maximum Number of Epochs	20
Stopping RMS Error	10^{-3}
Stopping Gradient Error	10^{-5}

Table 6.14 Test 3 Parameters

Sensor	Feature Extraction	Accuracy	Training epochs	Training Time (sec)	# of Failed Tests /144
Knock 1	AVG-CAD	100%	6	140	0
	AVG-CAD-FFT	98.6%	20	757	3
	AVG-CAD-WT	100%	7	166	0
Knock 2	AVG-CAD	100%	7	190	0
	AVG-CAD-FFT	99.3%	20	714	1
	AVG-CAD-WT	100%	7	150	0
Knock 3	AVG-CAD	100%	6	140	0
	AVG-CAD-FFT	100%	20	802	0
	AVG-CAD-WT	100%	6	127	0
Knock 4	AVG-CAD	100%	6	171	0
	AVG-CAD-FFT	70.1%	20	761	36
	AVG-CAD-WT	100%	7	152	0
Microphone 1	AVG-CAD	100%	7	213	0
	AVG-CAD-FFT	100%	5	172	0
	AVG-CAD-WT	100%	13	346	0
Accelerometer 1	AVG-CAD	100%	7	216	0
	AVG-CAD-FFT	97.2%	20	720	6
	AVG-CAD-WT	100%	5	121	0
Accelerometer 2	AVG-CAD	100%	11	382	0
	AVG-CAD-FFT	75%	20	590	34
	AVG-CAD-WT	100%	6	137	0

Table 6.15 Test 3 Results

The following figures present a sample of the obtained confusion matrices and the MSE graphs for Microphone 1:

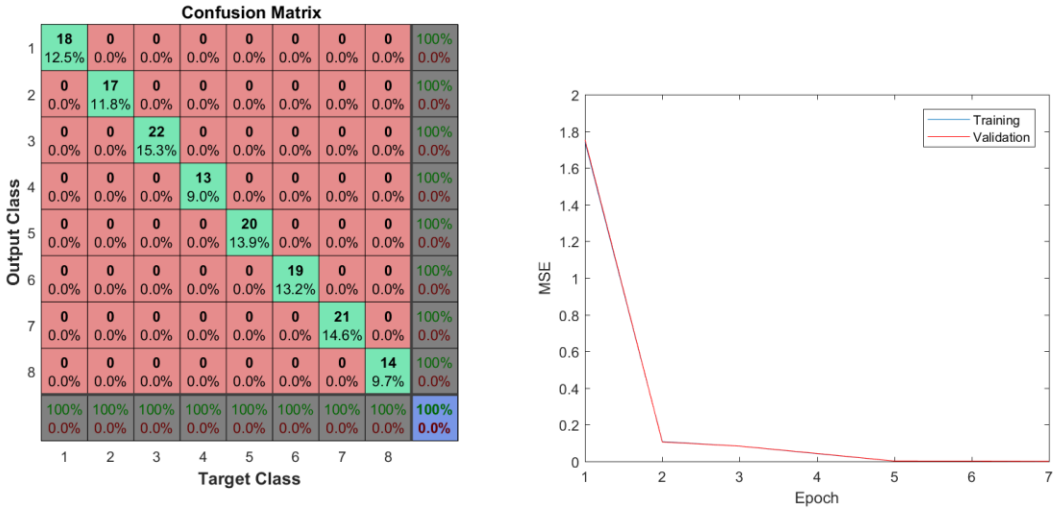


Figure 6.20 Microphone 1 Average-CAD LM Training Results

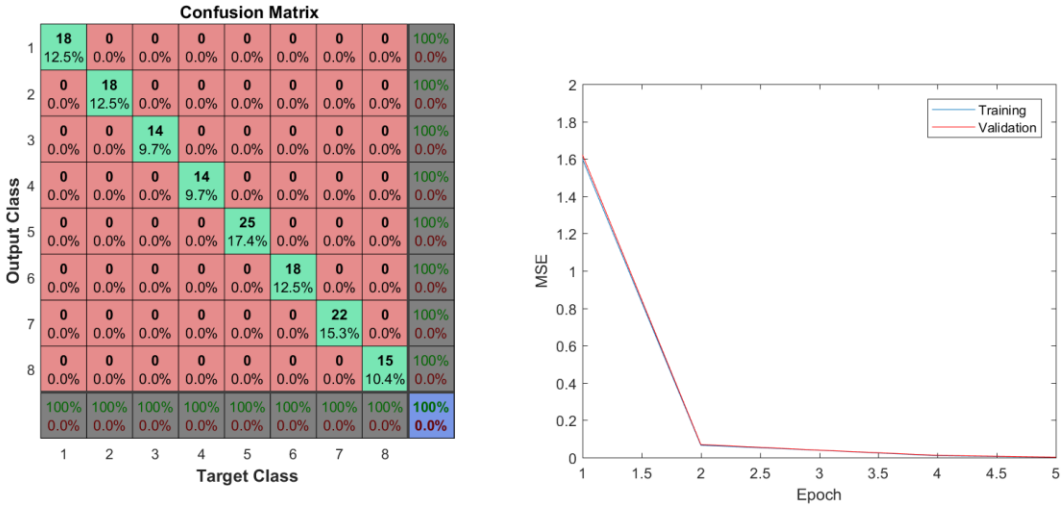


Figure 6.21 Microphone 1 Average-CAD-FFT LM Training Results

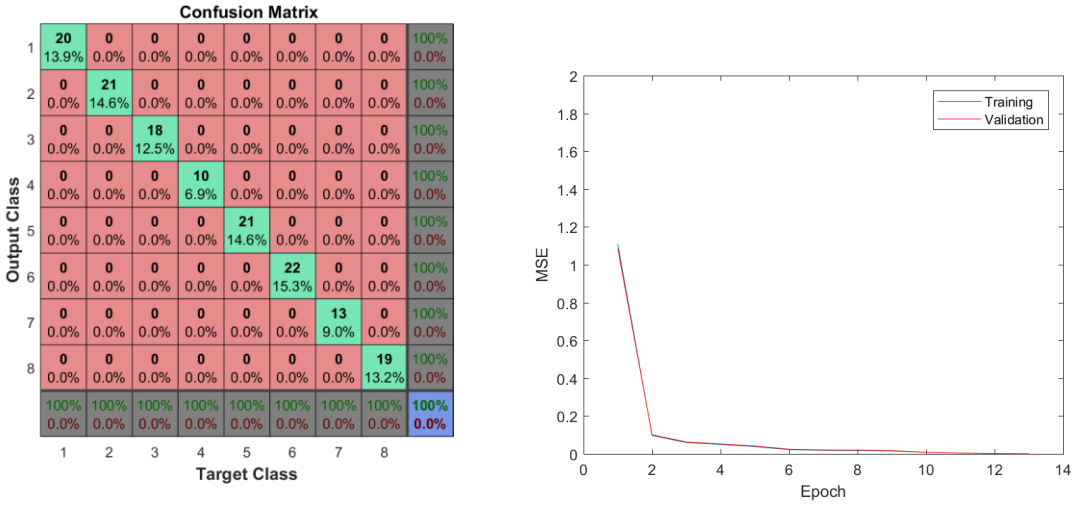


Figure 6.22 Microphone 1 Average-CAD-WT LM Training Results

Taking into consideration the results obtained by the three tests, the following 2 observations are derived:

- Similar to tests 1 and 2, Average-CAD-FFT had the poorest performance among the three feature extraction techniques in terms of accuracy, convergence rate, and training time.

- By analysing the sensors' results separately, it is clear that Knock 4 and accelerometer 2 had the lowest performance among all 7 sensors. This is highly linked to the mounting location and direction of these sensors. According to Figure 6.8, Accelerometer 2 is set in parallel with the piston motion of cylinder 1. On the other hand, Knock 4 which is close to cylinder 8 is diagonally the farthest sensor from the pre-ignition location at cylinder 1.

The results of this third test reiterate the conclusions drawn from the previous two tests regarding the Average-CAD-FFT feature extraction algorithm. What it more, this test proved that individual sensors can detect and classify faults with different intensities using the M1-Average-CAD FDD

technique. Finally, the above results highlight the importance of the mounting direction and location of the sensors on the engine block.

Summary

In order to assess the performance of the M1-Average-CAD FDD technique, three different tests on two engine setups were conduced. The results confirmed the effectiveness of the M1-Average-CAD signal processing method to detect single misfire, pre-ignition, and knocking fault conditions. However, this FDD technique was unable to classify dual misfire with high accuracies. To be able to achieve that, the combined contribution of all sensors is required. This sensor consolidation is implemented by M2-CAD-IEMSPCA, and the results of this second FDD technique are described in the following section.

6.3. M2-CAD-IEMSPCA FDD Results

The M2-CAD-IEMSPCA technique derives further information about the system by exploiting the inherent correlation between the logged sensors. This FDD technique generates the fault coefficients F_c, see Figure 3.6, which are used to train the MLP neural network classifier. In preparing the input vector for the network, there are two cases that were considered:

- Training the neural network using the F_c coefficients of each individual sensor separately. In this case, the number of trained neural networks is equal to the number of sensors that are connected to the FDD system. This allows to compare M2-CAD-IEMSPCA and M1-Average-CAD results.

- Training the neural network by combining all sensors' F_c coefficients in one input vector. There are two possible merging solutions described in Figure 6.23. In the first type, named 'Block Combination', the features from individual sensors are concatenated together to

form the neural network input vector. The second type, defined as 'Mixed Combination', merges the individual F_c coefficients one by one to generate the input vector to the network.

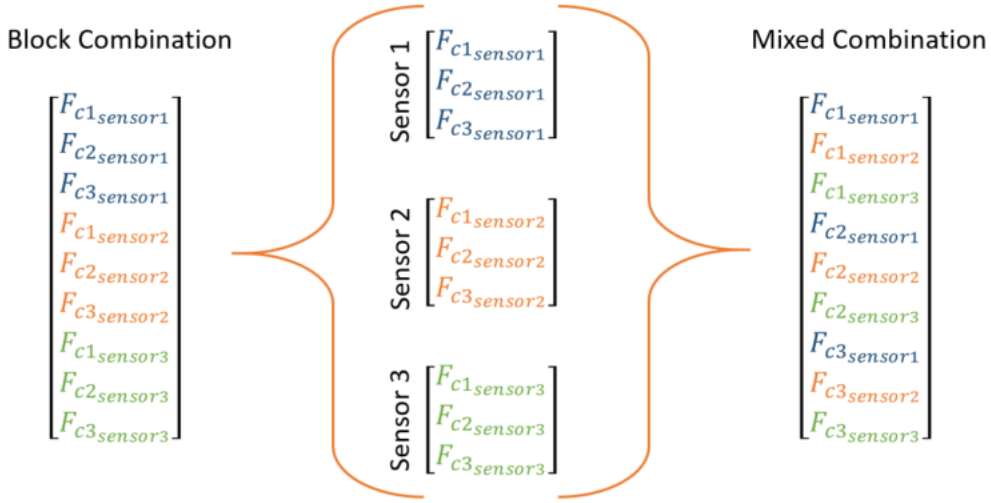


Figure 6.23 Combining F_c Coefficients

The following three subsections are the same tests that were implemented with the first FDD technique (M1-Average-CAD).

Test 1: Single Misfire Condition

The following Table 6.16 summarizes the fixed parameters that were predefined to generate the results of test 1. In addition to the network parameters, M2-CAD-IEMSPCA uses the Daubechies mother wavelet which was set with a scaling factor 'db16' and a depth level of 4. The maximum number of training epochs was initially set to 200. Due to the performance produced by the first classification results, another test was carried by allowing the maximum number of training epochs to reach 600.

Parameter	Value
Total Number of Features	36090
Ratio of Training : Validation : Testing	60% : 20% : 20%
Maximum Number of Epochs	200 or 600
Stopping RMS Error	10^{-3}
Stopping Gradient Error	10^{-5}
Mother wavelet	Daubechies
Wavelet Scaling Factor	db16
Wavelet Level	4

Table 6.16 Test 1 Parameters

Sensors	Accuracy	Training Epoch	Training Time (sec)	# of failed tests / 7218
Knock	48.6%	200	122	3658
Accelerometer 1	56.7%	200	130	3200
Accelerometer 2	54.6%	200	136	3267
Microphone 1	57.6%	200	117	3128
Microphone 2	70%	200	139	2212
Microphone 3	63.5%	200	132	2673
Block Combination	83.1%	200	2056	1166
Mixed Combination	80.7%	200	3154	1406

Table 6.17 Test 1 Results (200 epochs)

Sensors	Accuracy	Training Epoch	Training Time (sec)	# of failed tests / 7218
Knock	50.5%	297	204	3542
Accelerometer 1	57.3%	335	195	3113
Accelerometer 2	55.3%	266	189	3261
Microphone 1	60.5%	310	195	2791
Microphone 2	75.6%	493	290	1744
Microphone 3	64.1%	531	331	2590
Block Combination	86.7%	475	6342	923
Mixed Combination	85.5%	507	7689	1035

Table 6.18 Test 1 Results (600 epochs)

The subsequent figures present the confusion matrix and the MSE graph generated by “Block Combination” and “Mixed Combination” feature arrangements.

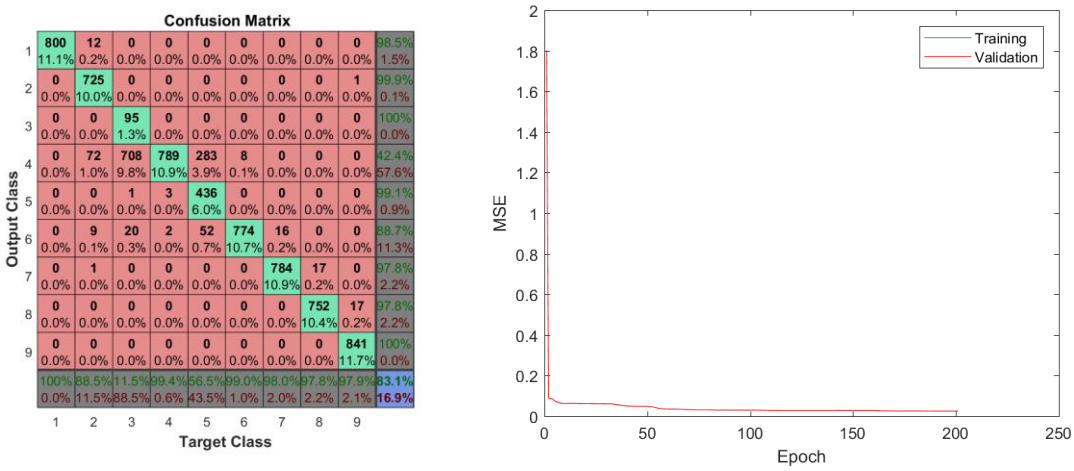


Figure 6.24 Block Combination Sensor Fusion (max 200 epochs)

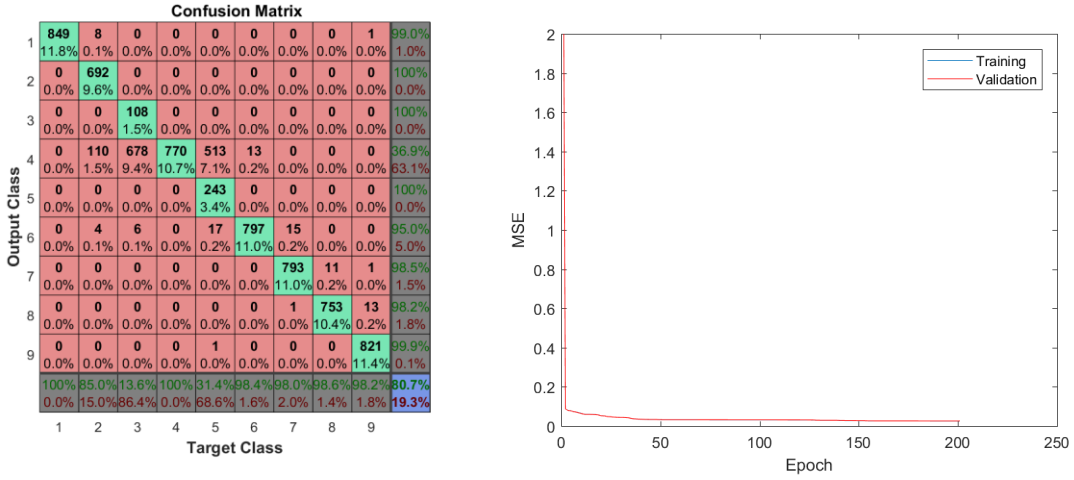


Figure 6.25 Mixed Combination Sensor Fusion (max 200 epochs)

From the above results, the following observations are asserted:

- When separate, the sensors were not capable of achieving good classification outcomes for detecting single cylinder misfire conditions. This is opposite to the M1-Average-CAD technique where accurate fault classification was achieved using the sensors' features independently.

- According to Table 6.17, the maximum number of training epochs was reached by all 8 MLP networks. That is why, this limit was increased to 600 to allow for more training. However, this parameter change barely improved the recorded performance.

- Combining the features together achieved better classification results than the stand alone sensors. In addition to that, the 'Block combination' arrangement recorded a better performance than the 'Mixed combination' arrangement.

- By analyzing the number of training epoch and the training time together, it is clear that M2-CAD-IEMSPCA's classification speed was higher than that of M1-Average-CAD technique.

- Microphone 1 had the poorest classification results which confirms what was discussed in the previous section.

- The performance of the two accelerometers was slightly better than that of the knock sensor.

M2-CAD-IEMSPCA results demonstrate that this technique is a powerful FDD tool especially when all sensors' features are combined together. Furthermore, the accuracy of this FDD method was barely affected by the increase in the number of training epochs.

Test 2: Double Misfire Fault Condition

In this test, M2-CAD-IEMSPCA is tuned based on the wavelet scaling factor. Three scaling factors were considered: db10, db16, and db20. The remaining parameters were kept fixed with respect to the first test.

Parameter	Value
Total Number of Features	45125
Ratio of Training : Validation : Testing	60% : 20% : 20%
Maximum Number of Epochs	200
Stopping RMS Error	10^{-3}
Stopping Gradient Error	10^{-5}
Mother wavelet	Daubechies
Wavelet Scaling Factor	db10, db16, db20
Wavelet Level	4

Table 6.19 Test 2 Parameters

Sensors	Debauchies	Accuracy	Training Epoch	Training Time	# of Failed tests / 9025
Knock	db10	33.9%	200	314	5878
	db16	35.3%	200	312	5766
	db20	34.7%	200	285	5923
Accelerometer 1	db10	42.1%	200	299	5234
	db16	40.1%	200	345	5307
	db20	42.2%	200	300	5272
Accelerometer 2	db10	85.8%	200	279	1605
	db16	81.5%	115	188	1760
	db20	80.6%	74	115	1712
Microphone 1	db10	50%	146	264	4570
	db16	48%	167	259	4750
	db20	46.6%	143	280	4879
Microphone 2	db10	50.8%	200	311	4443
	db16	51.3%	200	330	4413
	db20	53.1%	200	284	4258
Block Combination	db10	84.5%	76	1672	1458
	db16	90.9%	92	1743	833
	db20	82.8%	82	1482	1570
Mixed Combination	db10	83.7%	96	47933	1500
	db16	82.8%	105	3797	1593
	db20	81.4%	95	38471	1732

Table 6.20 Test 2 Results

The following figures are samples of the classification results for Accelerometer 2:

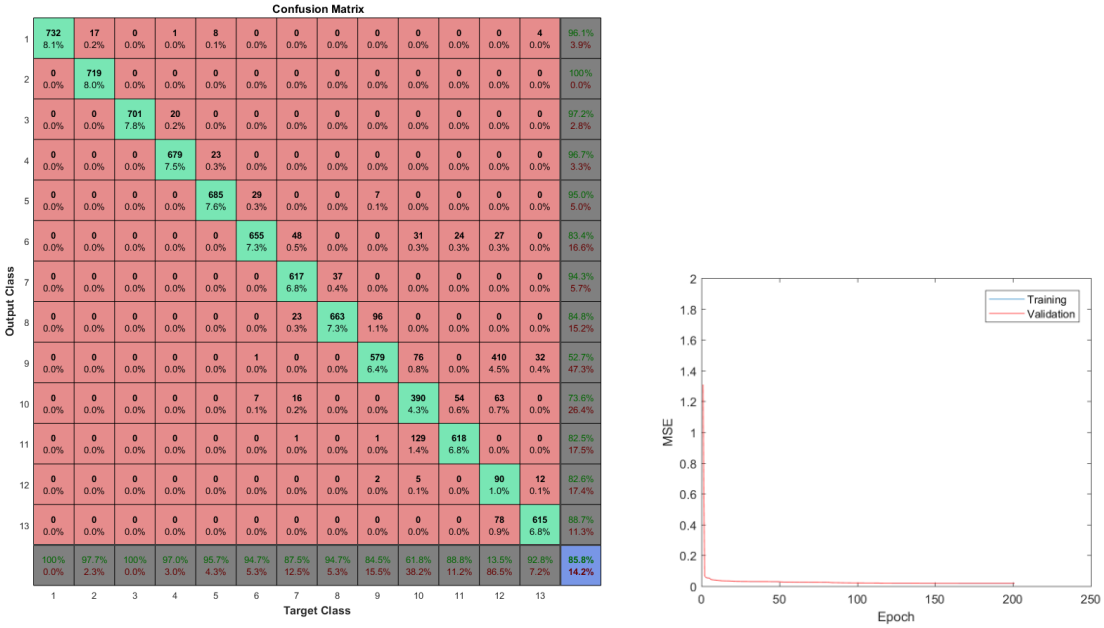


Figure 6.26 Accelerometer 2 with Scaling Factor (db10)

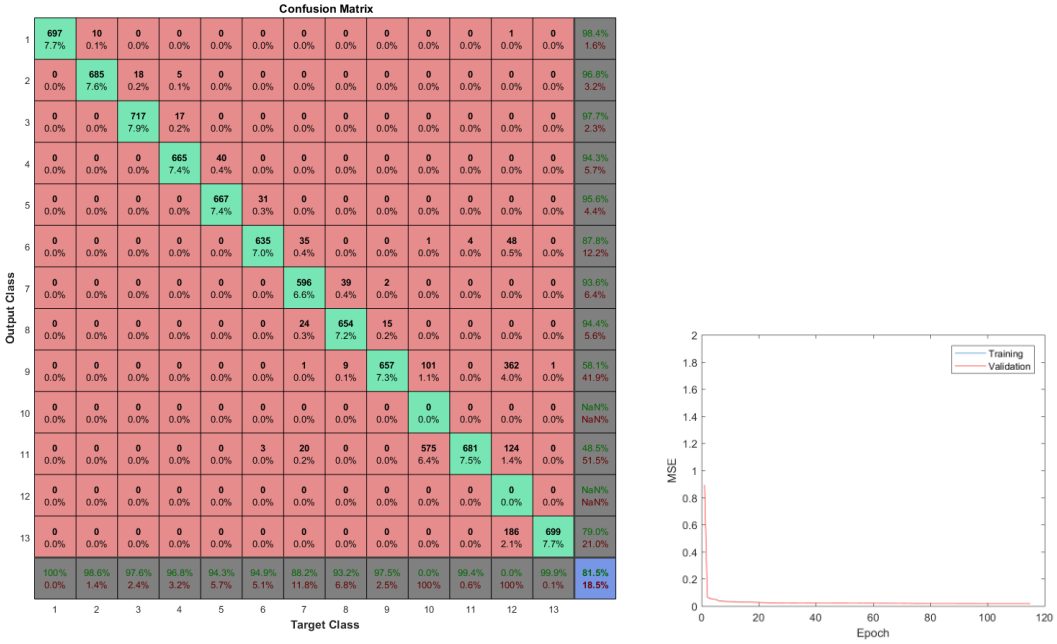


Figure 6.27 Accelerometer 2 with Scaling Factor (db16)

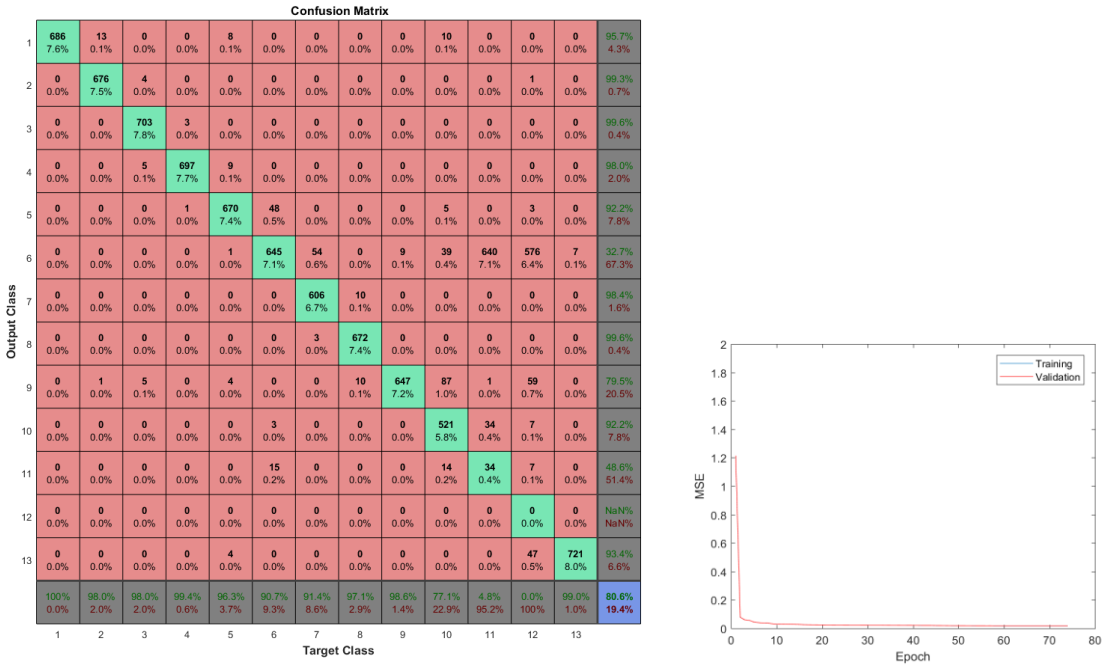


Figure 6.28 Accelerometer 2 with Scaling Factor (db20)

Based on the above results, the following observations are derived:

- The best performance was achieved by the 'Block Combination' arrangement when the scaling factor was set to db16. The maximum accuracy recorded was 90.9%.
- 'Block Combination' arrangement achieved better results than 'Mixed Combination' arrangement.
- The wavelet scaling factor 'db20' achieved the worst results among the tested scaling factors (db10, db16, and db20).
- Accelerometer 2 results were considerably better than the remaining sensors while achieving an accuracy average of around 81%.

Similar to test 1, changing the wavelet scaling factor had little effect on the overall performance of M2-CAD-IEMSPCA technique. For that reason, db16 was kept as a suitable scaling factor. Regardless, this FDD technique was able classify dual misfire conditions with an accuracy of 90.9%.

Test 3: Knock & Pre-ignition Fault Condition

In this third test, M2-CAD-IEMSPCA performance was monitored with respect to three wavelet levels: 4, 5, and 6. Similar to the previous two tests, the detection capability of the sensors (either alone or combined) is reported. In addition to that, this experiment considers two main cases:

- In the first case, faults of the same type, but with different levels of severity, are clustered together as one classification output category. Meaning, training the neural network is reduced to classifying only 3 categories: Knocking, Pre-ignition, and healthy conditions.

- In the second case, faults with different severities are considered as separate categories. This case is similar to the 3rd test that was previously implemented to assess the M1-Average-CAD FDD technique.

Parameter	Value
Total Number of Features	19920
Ratio of Training : Validation : Testing	60% : 20% : 20%
Maximum Number of Epochs	200
Stopping RMS Error	10^{-3}
Stopping Gradient Error	10^{-5}
Mother wavelet	Daubechies
Wavelet Scaling Factor	db16
Wavelet Level	4,5, and 6

Table 6.21 Test 3 Parameters

Faults Clustering	Sensors	Wavelet level	Accuracy	Training Epoch	Training Time (sec)	# of Failed Tests / 3984
Combined Faults	Knock 1	4	100%	139	7	0
		5	100%	66	4	0
		6	100%	18	4	0
	Knock 2	4	99.9%	95	5	1
		5	100%	90	5	0
		6	100%	25	5	0
	Knock 3	4	99.9%	119	6	0
		5	100%	49	3	0
		6	100%	24	5	0
	Knock 4	4	99.9%	151	7	0
		5	100%	136	6	0
		6	100%	19	10	0
	Microphone 1	4	99.9%	101	5	0
		5	100%	48	3	0
		6	100%	40	16	0
	Accelerometer 1	4	100%	52	4	0
		5	100%	39	3	0
		6	100%	17	6	0
	Accelerometer 2	4	100%	200	9	0
		5	100%	92	5	0
		6	100%	68	17	0
	Block Combination	4	100%	6	10	0
		5	100%	6	15	0
		6	100%	5	43	0
	Mixed Combination	4	100%	8	14	0
		5	100%	7	17	0
		6	100%	6	52	0
Separate Faults	Knock 1	4	79%	200	59	848
		5	97.4%	154	34	27
		6	99.9%	196	122	2
	Knock 2	4	84%	109	36	659
		5	98%	113	27	22
		6	98.5%	184	119	22
	Knock 3	4	78%	125	47	897
		5	98.5%	118	29	29
		6	100%	166	107	0
	Knock 4	4	83%	113	40	649
		5	98.7%	114	27	23
		6	100%	159	103	0
	Microphone 1	4	89%	200	63	438
		5	98.3%	200	46	19
		6	100%	123	80	0
	Accelerometer 1	4	79%	200	62	810
		5	95.4%	183	43	50

		6	99.4%	200	130	11
	Accelerometer 2	4	86%	200	64	546
		5	92.6%	200	47	109
		6	100%	200	130	0
	Block Combination	4	100%	35	318	0
		5	100%	24	367	0
		6	100%	28	1502	0
	Mixed Combination	4	100%	30	269	0
		5	100%	21	309	0
		6	100%	18	951	0

Table 6.22 Test 3 Results

The following figures are sample of the classification results produced by Accelerometer 1:

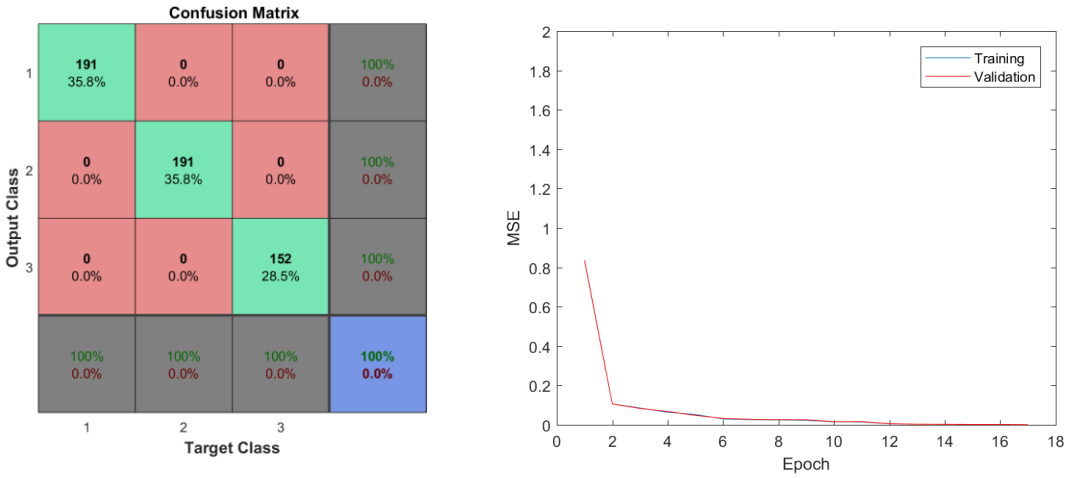


Figure 6.29 Accelerometer 1 Classification Results

(Combined Faults, Wavelet level 6)

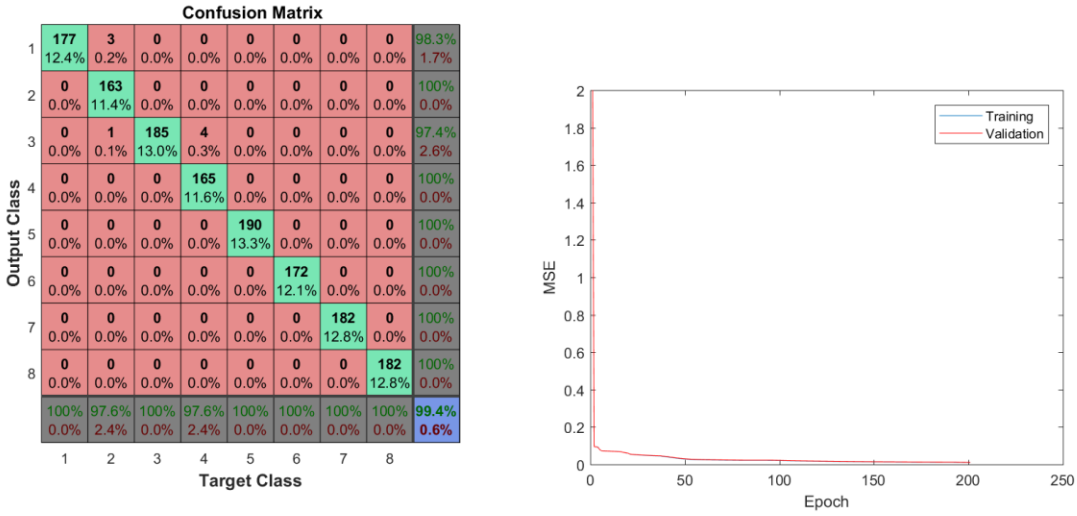


Figure 6.30 Accelerometer 1 Classification Results

(Separate Faults, Wavelet level 6)

By examining the above results, the following observations are stipulated:

- Increasing the wavelet level improved the classification capabilities of M2-CAD-IEMSPCA. For that reason, wavelet level 6 should be used with this FDD technique.
- The training times achieved by M2-CAD-IEMSPCA were shorter than those obtained by M1-Average-CAD.
- Combining faults with different severities did not confuse the network and its training was successfully achieved.
- Setting the wavelet level at 6 allowed M2-CAD-IEMSPCA to record similar classification accuracies to those obtained by the M1-Average-CAD FDD technique. This was the case even when the sensors' features were used separately for the training.

- M2-CAD-IEMSPCA recorded similar performance across all sensors. This is opposed to M1-Average-CAD where knock 4 (farthest from the cylinder 1) and accelerometer 2 (parallel to the piston movement) had the lowest performances due their mounting location and direction respectively. With that, M2-CAD-IEMSPCA was proven to be a multivariate FDD technique.

M2-CAD-IEMSPCA successfully detected all the faults and classified them accordingly when the wavelet level is 6. Treating the faults separately or combining them together had no effect on the diagnostics capabilities of this technique. Finally, the training time can be reduced by cascading two MLP topologies, one for classifying the faults and the other for classifying the severities within each fault category.

Summary

To improve the performance of the M2-CAD-IEMSPCA, three attempts were made. In the first test, the maximum number of training epochs was increased. In the second test, the wavelet scaling factor was modified. In the last test, the wavelet level was increased from 4 to 6. These three tests clearly showed that the results achieved by M2-CAD-IEMSPCA are highly dependent on the wavelet level. This being said, this second FDD method is a multivariate technique that requires using all the sensors together for maximum reliability.

6.4. M3-CASA FDD Results

This third and last FDD technique is tailored to misfire fault detection. It requires at least two synchronization signals to extract the engine cycles. In this work, both the camshaft and the crankshaft position sensors were sampled. According to Figure 6.31, the camshaft signal (top graph) is used to distinguish the right from the left cylinder bank, and it is used to localize the beginning of a new engine cycle. As shown in the below figure, 5 complete cycles (2 revolutions

of crank = 720°) are extracted from this recording. The next step is to isolate the contribution of each cylinder, as described by Figure 3.9 using equations (3.17) and (3.18).

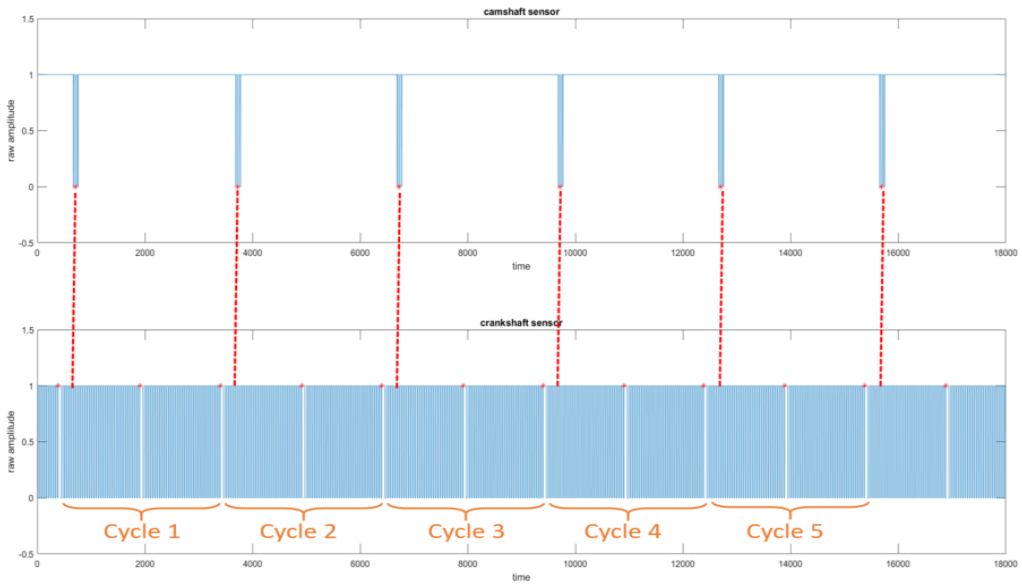


Figure 6.31 Synchronization Between Camshaft and Crankshaft

Figure 6.32 and Figure 6.33 plot the time intervals that represent each cylinder in crank angle domain. To detect misfire, the time difference between consecutive cylinders is computed. If the difference is above a fixed threshold, which is specified in Table 6.23, the misfire condition is detected and localized. The misfire threshold is a function of three variable: engine speed, engine load, and sampling frequency. Table 6.23 summarizes the two thresholds that were used in the subsequent tests to detect single and dual misfiring conditions. This being said, the threshold used to flag double misfire is lower than the threshold used to detect single misfire. This allows to detect two misfiring cylinders that are successive to one another in the engine's firing order.

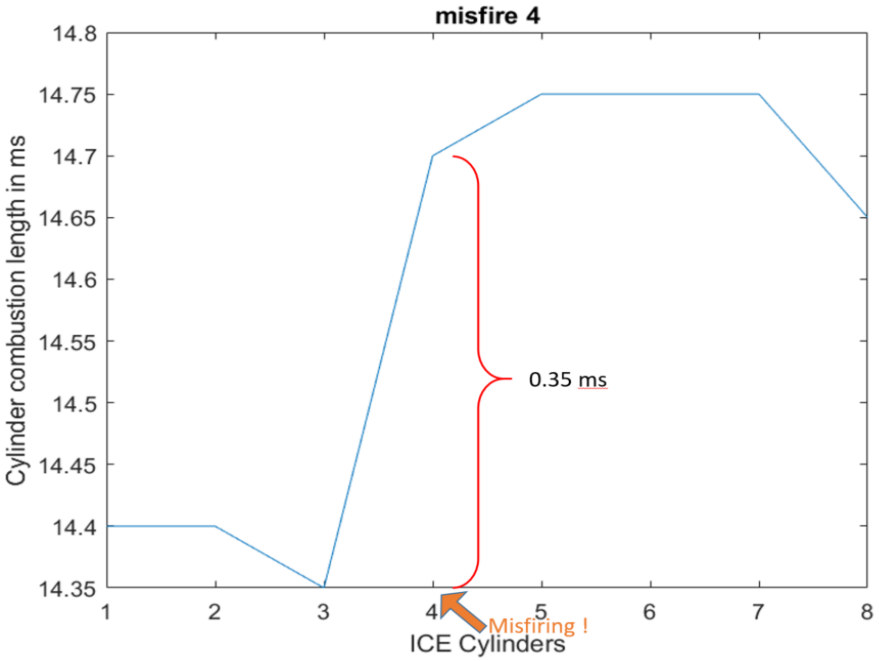


Figure 6.32 Misfire Cylinder 4

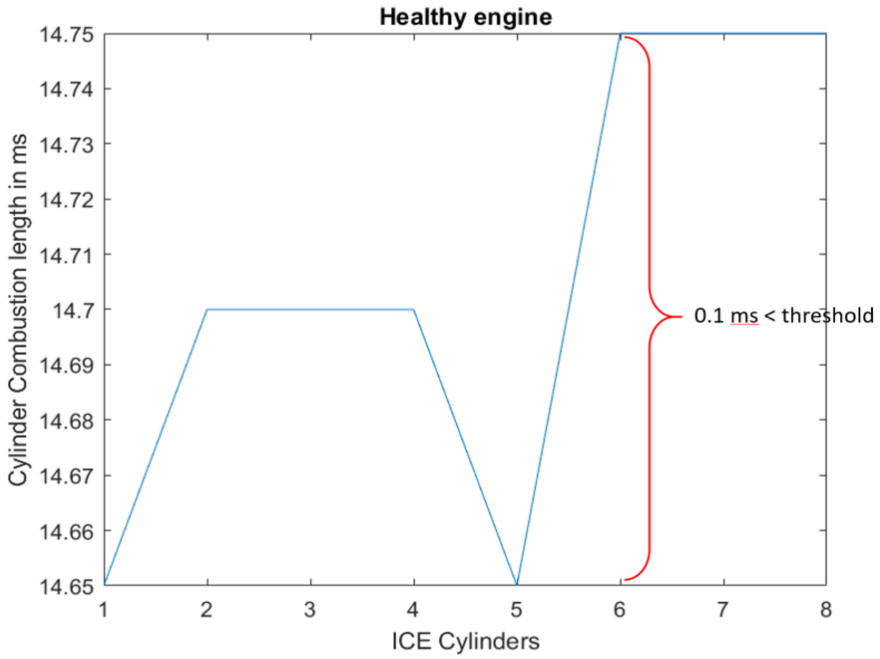


Figure 6.33 Healthy Engine

Threshold Parameters (800 rpm, 20 kHz)	Values (ms)
Single Misfire Threshold	0.25
Dual Misfire Threshold	0.2

Table 6.23 M3-CASA FDD Parameters

Test 1: Single Misfire Condition

Test 1 was carried on the Ford Crown Victoria Car. In this test a total of 4275 cycles (2475 healthy + 225 faulty X 8 cylinders) were used to asses the FDD capabilities of the M3-CASA technique. Unlike the previous two FDD technique, the detection is carried without using neural networks. Table 6.24 summarizes the results of the experiment.

Condition	Accuracy	Average Time (sec)
Cylinder Misfire 1	100%	0.097444
Cylinder Misfire 2	100%	0.099070
Cylinder Misfire 3	100%	0.096045
Cylinder Misfire 4	100%	0.096424
Cylinder Misfire 5	100%	0.093456
Cylinder Misfire 6	100%	0.092875
Cylinder Misfire 7	100%	0.093771
Cylinder Misfire 8	100%	0.093645
Healthy	100%	0.106125

Table 6.24 M3-CASA Test 1 Results

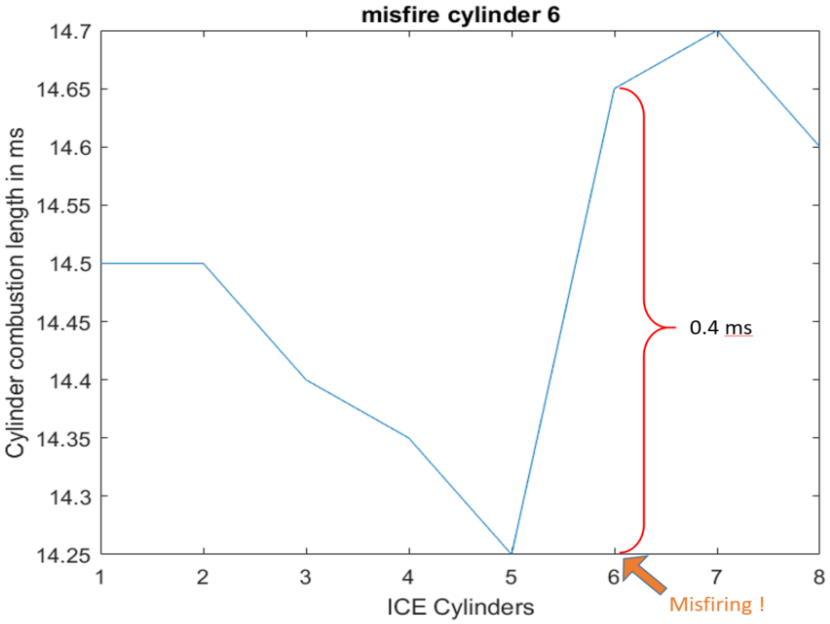


Figure 6.34 Misfire Cylinder 6

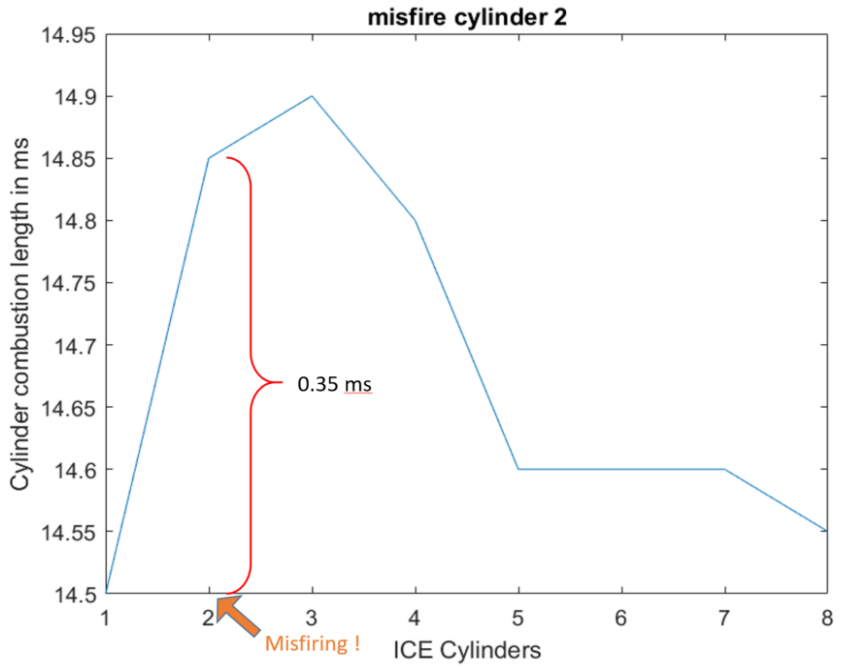


Figure 6.35 Misfire Cylinder 2

As shown in the above table, M3-CASA FDD technique achieve 100% accuracy with processing times less than a second. This recorded performance exceeds both M2-CAD-IEMSPCA and M1-Average-CAD in all aspects.

Test 2: Double Misfire Fault Condition

In this second test, 4 dual misfire conditions was simulated on the Ford Crown Victoria Engine. A total of 2025 engine cycles (1125 healthy + 225 X 4 dual misfire) were collected to asses the detection capabilities of M3-CASA FDD technique with dual misfire conditions. Table 6.25 summarizes the findings of the experiment.

Condition	Accuracy	Average Time
Cylinder Misfire 1 & 7	100%	0.111434
Cylinder Misfire 2 & 8	100%	0.110169
Cylinder Misfire 3 & 5	100%	0.112701
Cylinder Misfire 4 & 6	100%	0.103929
Healthy	100%	0.106778

Table 6.25 M3-CASA Test 2 Results

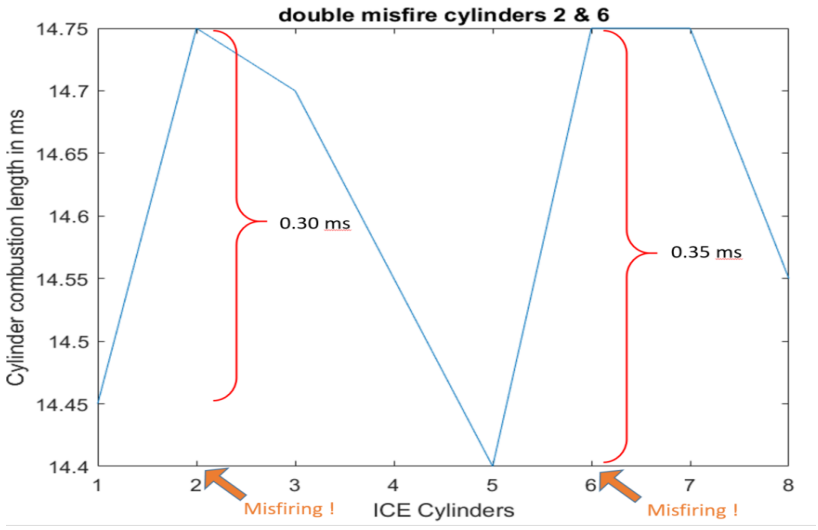


Figure 6.36 Cylinder Misfire 2 and 6

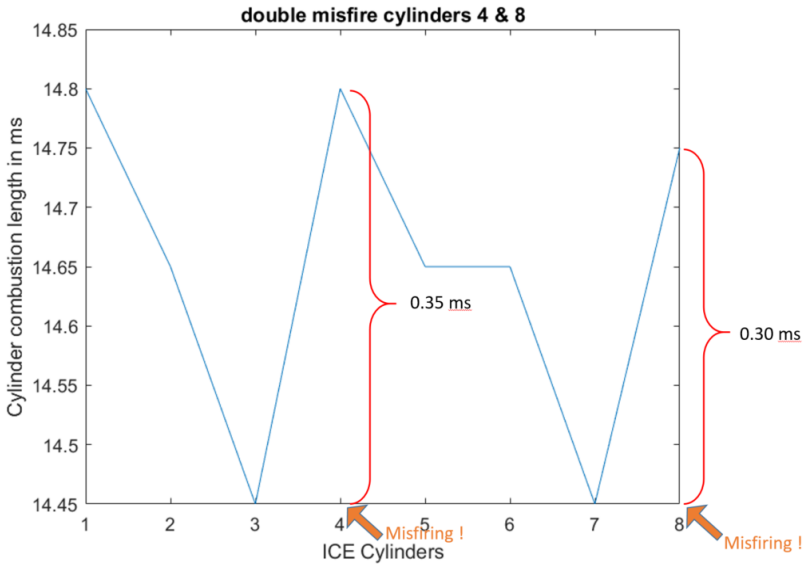


Figure 6.37 Cylinder Misfire 4 and 8

Similar to the first test, the accuracy and speed of the M3-CASA FDD technique were better than those recorded by the other 2 FDD techniques. Detecting dual misfire conditions was also achieved for successive cylinders in engine's firing order.

Summary

The M3-CASA FDD technique was able to detect single-, and double- cylinder misfire conditions quickly with 100% accuracy. This technique does not involve training a neural network. However, it requires defining the threshold limits that depend on the engine speed and load as well as the sampling frequency of the data acquisition system. For that, tuning these parameters initially is a necessary step to use the M3-CASA FDD technique as a reliable tool to detect misfire in Internal Combustion Engine.

6.5. Summary

This chapter described the two experimental setups of this work. They were used to test the 3 FDD techniques discussed in Chapter 3, namely M1-Average-CAD, M2-CAD-IEMSPCA, and M3-CASA. Three different tests were conducted.

- Test 1 and Test 2: misfire fault conditions (Setup 1: Ford Crown Victoria engine)
- Test 3: knocking and pre-ignition (Setup 2: Ford Coyote engine)

The tests' results were used to compare the three FDD techniques: although the M3-CASA FDD method detected misfire faults accurately and rapidly, it lacked the capability to detect faults occurring across all cylinders, such as knocking. On the other hand, M2-CAD-IEMSPCA and M1-Average-CAD techniques had comparable classification accuracies. Finally, M2-CAD-IEMSPCA was faster in classifying the faults, whereas M1-Average-CAD was faster to generate the features needed for training.

Chapter 7. Conclusions and Future Work

Fault Detection and Diagnosis is an important consideration for the operation of Internal Combustion Engines (ICEs). This work presents the development of a FDD data acquisition system for ICEs. To perform engine diagnostics using this device, three different FDD strategies were considered. Furthermore, their diagnostic capabilities were experimentally verified against three faults conditions: misfire, pre-ignition, and knocking.

The following are the contributions of this work:

1. The development of all the hardware components that make up the FDD system. The proposed device integrates a microcontroller to sample a number of sensors and a microprocessor to analyse the data onboard. The sensors' signal conditioning circuits were developed and prototyped on a compact printed circuit board (PCB). The FDD device sits inside the car, and it is powered by the vehicle's battery.

2. The development of the firmware that allows the FDD system to sample sound and vibration sensors at a maximum rate of 125 kHz. The firmware implements the SPI protocol to configure and control an external ADC module. The data transfer between the microprocessor and the microcontroller was implemented using Termios library for maximum bandwidth. The firmware of the FDD device is user friendly and open source.

3. The Industrial Extended Multi-Scale Principal Component Analysis (IEMSPCA) algorithm was revised for crank-angle domain processing that is more applicable to ICE. The M2-CAD-IEMSPCA FDD technique was developed, and it incorporates two additional steps: Crank-Angle Domain (CAD) transform and a moving average (LPF) for the generated F_c coefficients. Furthermore, the fault detection capabilities of this new technique were compared with the M1-Average-CAD FDD method.

4. The development of the Crank-Angle Speed Analysis (M3-CASA) FDD technique that flawlessly detected all simulated misfire fault conditions. This FDD technique had the fastest processing speed compared to the others methods in this research.

This research also highlighted the importance of both the mounting location and the direction of the *vibro-acoustic* transducers when placed on the engine. The limitations coming from the sensors' arrangement were mitigated using multi-variate techniques, such as M2-CAD-IEMSPCA. Finally, this work validated the reliability of the knock sensors to detect other defects than knocking fault conditions.

Future Work

Both the FDD device and the FDD techniques can benefit from the following improvements:

- Combining all FDD system's hardware elements in one compact design. This makes the device robust against RF interference (from exposed wires) and vibration. Accordingly, the new PCB prototype should include the accelerometer drivers and the switching buck boost converter.

- Modifying the firmware and the source codes of the teensy microcontroller to sample simultaneously 8 analog channels with rates above 100 kHz.

- Conducting the sensitivity analysis on all of M1-Average-CAD, M2-CAD-IEMSPCA, and M3-CASA FDD techniques. The analysis should tune the parameters of the aforementioned FDD techniques in order to detect faults at different engine speeds and loads for any practical application to be considered.

Once all the parameters for the FDD techniques are tuned, the succeeding designs of the FDD system should adopt Application-Specific Instruction Set (ASIP) processors. Such types of

processing units will be optimized for the Internal Combustion Engine applications, so they reduce the size, the power consumption, and the total price of the FDD device.

Bibliography

[1] Encyclopedia Briatnnica , "History of Technology: Internal Combustion Engines," [Online]. Available: https://www.britannica.com/technology/history-of-technology/The-Industrial-Revolution-1750-1900#ref10451.

[2] Penninger Ankal, Lezsovits Ferenc, Rohaly Janos, Wolff Vilmos, and Bereczky Akos, "Internal Combustion Engines (Heat Engines II)," Technical University of Budapest, Budapest, 2006.

[3] Belgele from autoeng.cu, "4-stroke Diesel Engine," [Online]. Available: http://autoeng.cu.edu.tr/tr/Belgeler%5Cduyuru%5C4-stroke%20Diesel%20Engine.pdf.

[4] Caltech.edu, "Internal Combustion Engines," 2013. [Online]. Available: https://authors.library.caltech.edu/25069/6/AirPollution88-Ch4.pdf.

[5] Jim Smart from Truck Trend Network, "Undertstanding the Ford 4.6L/5.4L 3V SOHC V8," 13 Oct. 2017. [Online]. Available: http://www.trucktrend.com/features/1710-undertstanding-the-ford-4-6l5-4l-3v-sohc-v8/.

[6] DieselNet and Emission Standards, "United States: Light-Duty Vehicles: GHG Emissions & Fuel Economy," Aug. 2013. [Online]. Available: https://www.dieselnet.com/standards/us/fe_ghg.php.

[7] Hodgins Sean, "A Wireless Sensor for Fault Detection and Diagnosis of Internal Combustion Engines," McMaster University, Hamilton, Ontario, Canada, 2016.

[8] Feng Yifei, "FAULT DETECTION AND DIAGNOSIS OF AN INTERNAL COMBUSTION ENGINE," McMaster University, Hamilton, Ontario, 2016.

[9] Ismail Mahmoud, Industrial Extended Multi-Scale Principle Components Analysis for Fault Detection and Diagnosis of Car Alternators and Starters, Hamilton: McMaster University, 2015.

[10] Joe Qin S., "Data-driven Fault Detection and Diagnosis for Complex Industrial Processes," *ELSEVIER*, vol. 42, no. 8, pp. 1115-1125, 2009.

[11] Gertler Janos, Fault Detection and Diagnosis in Engineering Systems, Basel, New York: CRS Press, May 15, 1998.

[12] Venkatasubramanian Venkat, Rengaswamy Raghunathan, Yin Kewen, Kavuri, and Surya N., "A Review of Process Fault Detection and Diagnosis Part 1: Quantitative Model-based Methods," *ELSEVIER Computers and Chemical Engineering*, vol. 27, no. 3, pp. 293-311, Mar., 15, 2003.

[13] Mehra R. K. and Peschon J., "An Innovation Approach to Fault Detection and Diagnosis in Dynamic Systems.," *Automatica,* vol. 7, pp. 637-640, 1971.

[14] Willsky A. S., "A Survey of Design Methods for Failure Detection in Dynamic Systems," *Automatica,* vol. 12, pp. 601-611, 1976.

[15] Willsky A. S. and Jones H. L., "A Generalized Likelihood Ratio Approach to the Detection and Estimation of Jumps in Linear Systems," *IEEE Transactions on Automatic Control,* vol. 21, pp. 108-112, 1976.

[16] Himmelblau D. M., "Fault Detection and Diagnosis in Chemical and Petrochemical Processes," *Amsterdam: Elsevier press.,* 1978.

[17] Frank P. M. and Wunnenberg J. , "Robust Fault Diagnosis Unsing Unknown Input Observer Schemes," in *Fault Diagnosis in Daynamic Systems: Theory and Applications*, New York: Prentice Hall, R.J. Patton, P. M. Frank & R. N. Clark (Eds.), 1989.

[18] Insermann R., "Process Fault Detection Based on Modelling and Estimation Methods - A Survey," *Automatica,* vol. 20, no. 4, pp. 387-404, 1984.

[19] Chow E. and Willsky A., "Analytical Redundancy and the Design of Robust Failure Detection Systems," *IEEE Transactions on Automatic Control,* vol. 29, no. 7, pp. 603-614, 1984.

[20] Venkatasubramanian V. and Rich S. H., "An Object-oriented Tow-tier Architecture for Intergrating Compiled and Deep-level Knowledge for Process Diagnosis," *Elesvier: Computer and Chemical Engineering,* vol. 12, no. 9-10, pp. 903-921, 1988.

[21] Charniak D. and McDermott, E., Introduction to Artificial Intelligence, Massechusetts: Addison-Wesley Publishing Company, 1984.

[22] Hoskins Kaliyur and Himmelblau D., "Fault Diagnosis in Complex Chemical Plants Using Artificial Neural Networks," *American Institute of Chemical Engineers Journal,* vol. 37, no. a, pp. 137-141, 1991.

[23] Hotelling H., "Multivariate Quality Control Illustrated by the Testing of Sample Bombsights," in *Selected Techniques of Statistical Analysis*, New York, McGraw-Hill, 1947, pp. 113-184.

[24] Ugar, Powel, and Kemans, "Adaptive Neworks for Fault Diagnosis and Process Control," *Elesevier: Computers and Chemical Engineering,* vol. 14, no. 4-5, pp. 561-572, 1990.

[25] VenkatasubramannianV. and Chan K., "A Neural Network Mehodology for Process Fault Diagnosis," *American Institute of Chamical Engineers Journal,* vol. 35, no. 12, pp. 1993-2002, 1989.

[26] Venkatasubramanian Venkat, Rengaswamy Raghunathan, and Kavuri Surya N., "A Review of Process Fault Detection and Diagnosis Part 2: Qualitative Models and Search Strategies," *Elsevier: Computers & Chamical Engineering,* vol. 27, pp. 313-326, 2003.

[27] Venkatasubramanian Venkat, Rengaswamy Raghunathan, Kavuri Surya N., and Yin Kewen, "A Review of Process Fault Detection and Diagnosis Part 3: Process History Based methods," *Elsevier: Computers and Chamical Engineering,* vol. 27, pp. 327-346, 2003.

[28] Uzun S., Amira A., and Bouridane A., "FPGA implementations of fast Fourier transforms for real-time signal and image processing," *IET,* vol. 152, no. 3, pp. 283-296, 2005.

[29] Niederer Guido, Herzig Hans Peter, Shamir Joseph, Thiele Hans, Schnieper Marc, and Zschokke Christian, "Tunable, oblique incidence resonant grating filter for telecommunications," *Applied Optics,* vol. 43, no. 8, pp. 1683-1694, 2004.

[30] Feldm Joel, "Using the Fourier Transform to Solve PDEs," 21 Feb. 2007. [Online]. Available: https://www.math.ubc.ca/~feldman/m267/pdeft.pdf.

[31] Cooley James W. and Tukey John W., "An Algorithm for the Mchine Calculation of Complex Fourier Series," *Mathematics of Computation,* vol. 19, no. 90, pp. 297-301, 1965.

[32] Loan Charles Van, Computational Frameworks for the Fast Fourier Transform, Ithaca, New York: Frontiers in Applied Mathematics, 1992.

[33] Kar Chinmaya and Mohanty A. R., "Vibration and Current Transient Monitoring for Gearbox Fault Detection Using Multiresolution Fourier Transform," *Elsevier: Journal of Sound and Vibration,* vol. 311, pp. 109-132, 2008.

[34] Rai K. and Mohanty A. R., "Bearing Fault Diagnosis using FFT of Intrinsic Mode Functions in Hilbert-Huang Transform," *Elsevier: Mechanical Systems and Signal Processing,* vol. 21, pp. 2607-2615, 2007.

[35] Wu Shuanglong, Zuo Shuguang, Wu Xudong, Lin Fu, Zhong Hongmin, and Zhang Yaodan, "Vibroacoustic Prediction and Mechanism Analysis of Claw Pole Alternators," *IEE Transactions on Inductrial Electronics,* vol. 64, no. 6, pp. 4463-4473, 2017.

[36] Liu Chun-Lin, "A Tutorial of the Wavelet Transform," 23 feb. 2010. [Online]. Available: http://disp.ee.ntu.edu.tw/tutorial/WaveletTutorial.pdf.

[37] SHUKLA Panchamkumar D, "COMPLEX WAVELET TRANSFORMS AND THEIR APPLICATIONS (Master of Philosophy)," University of Strathclyde Glasgow, Scotland , United Kingdom, 2003.

[38] National Intruments, "Discrete Wavelet Transform (Advanced Processing Toolkit)," 2014. [Online]. Available: http://zone.ni.com/reference/en-XX/help/372656C-01/lvasptconcepts/wa_dwt/.

[39] Wu Jian-Da and Liu Chiu-Hong, "Investigation of engine fault diagnosis using discrete wavelet transform and neural network," *Elsevier: Expert Systems with Applications,* vol. 35, pp. 1200-1213, 2008.

[40] Kozionov Alexey, Kalinkin Mikhail, Natekin Alexey, and Loginov Alexander, "Wavelet-Based Sensor Validation: Differentiating Abrupt Sensor Faults From System Dynamics," *IEEE 7th International Symposium on Intelligent Signal Processing,* 2011.

[41] Lin Jing and Qu Liangsheng, "Feature Extraction Based on Morlet Wavelet and its Application for Mechanical Fault Diagnosis," *Journal of Sound and Vibration,* vol. 1, no. 234, pp. 135-148, 2000.

[42] Wu Jian-Da and Liu Chiu-Hong, "An Expert System for Fault Diagnosis in Internal Combustion Engines using Wavelet Packet Transform and Neural Network," *ELSEVIER: Expert Systems with Applications,* vol. 36, pp. 4278-4286, 2009.

[43] Wang Yi, Xu Guanghua, Liang Lin, and Jiang Kuosheng, "Detection of Weak Transient Signals Based on Wavelet Packet Transform and Manifold Learning for Rolling Element Bearing Fault Diagnosis," *ELSEVIER: Mechanical Systems and Signal Processing,* Vols. 54-55, pp. 259-276, 2015.

[44] Zarei Jafar and Poshtan Javad, "Bearing Fault Detection Using Wavelet Packet Transform of Induction Motor Stator Current," *ELSEVIER:,* vol. 40, pp. 763-769, 2007.

[45] Cauchy Al, "Sur l'equation ´ a l'aide de laquelle on determine les in ´ egalit ´ es s ´ eculaires des mouvements des planetes," *Oeuvres Completes (IIeme Serie),* vol. 9, 1829.

[46] Pearson K., "On lines and planes of closest fit to systems of points in space," *Philos Mag,* no. 6, p. 559–572, 1901.

[47] Hotelling H., "Analysis of a complex of statistical variables into principal components," *J Educ Psychol,* no. 25, pp. 417-441, 1933.

[48] Abdi Herve and Williams Lynne J., "Principal component analysis," in *WIREs Computational Statistics*, New york, John Wiley & Son s, In c, 2010, pp. 433-459.

[49] Jehan Tristan, "Creating music by listening (PhD thesis)," Massachusetts Institute of Technology, Massachusetts, Boston, 2005.

[50] Qin Joe S., "Statistical process monitoring: basics and beyond," *Journal of chemometrics,* vol. 17, no. 8-9, pp. 480-502, 2003.

[51] Yue H. Henry and Qin S. Joe, "Reconstruction-Based Fault Identification Using a Combined Index," *American Chemical Society*, no. 40, pp. 4403-4414, 2001.

[52] Jackson J., A User's guide to principal components., New York: John Wiley and Sons Inc., 1991.

[53] George Box, "Some theorems on quadratic forms applied in the study of analysis of variance problems, i. effect of inequality of variance in the one-way classification," *The Annals of Mathematical Statistics*, vol. 2, no. 25, pp. 290-302, 1954.

[54] Russel Evan L., Chiang Leo H., and Braatz Richard D., "Fault Detection in Industrial Processes Using Canonical Variate Analysis and Dynamic Principal Component Analysis," *Elsevier: Cheomometrics and Intelligent Laboratory Systems*, no. 51, pp. 81- 93, 2000.

[55] Cho Ji-Hoon, Lee Jong-Min, Choi Sang Wook, Lee Dongkwon, and Lee In-Beum, "Sensor Fault Identification Based on Kernal Principal Component Analysis," *IEEE*, pp. 1223-1228, 2004.

[56] Bakshi Bhavik R., "Multiscale PCA with Application to Multivariate Statistical Process Monitoring," *AIChE Journal*, vol. 7, no. 44, pp. 1596-1610, 1998.

[57] Yoon Seongkyu and MacGregor Jhon F., "Principal-Component Analysis of Multiscale Data for Process Monitoring and Fault Diagnosis," *AIChE Journal*, vol. 50, no. 11, pp. 2891-2903, 2004.

[58] Dunia R. and Qin J., "A Unified Geometric Approach to process and Sensor Fault Identification and Reconstruction: the Unidimensional Fault Case," *Computers and Chemical Engineering*, vol. 22, no. 7, pp. 927-943, 1998.

[59] Lachouri A., Baiche K., Djeghader R., and Doghmane N. Ouhtati, "Analyze and Fault Diagnosis by Multi-scale PCA.," *IEEE: Information and Communication Technologies: From Theory to Applications (ICTTA)*, pp. 1-6, 2008.

[60] Wenying Chen, "Application of Multi-scale Principal Component Analysis and SVM to the motor fault diagnosis.," *Information Technology and Applications*, vol. 3, pp. 131-134, 2009.

[61] Haykin Simon, Neural Networks and Learning Machines, New jersey: Pearson Education, Inc., 2009.

[62] MissingLink.ai, "Neural Network Concepts: 7 Types of Neural Network Activation Fucntions: How to Choose?," [Online]. Available: https://missinglink.ai/guides/neural-network-concepts/7-types-neural-network-activation-functions-right/.

[63] Samanta B., "Gear Fault Detecting Using Artifician Nerual Networks and Support Vector Machines with Genetic Algorithm," *Elsevier: Mechanical Systems and Signal Processing*, vol. 18, pp. 625-644, 2004.

[64] Jack L.B. and Nandi A. K., "Fault Detection Using Support Vector Machines and Artificial Neural Networks, Augmented by Genetic Algorithms," *Elsevier: Mechanical Systems and Signal Processing*, vol. 16, no. 2-3, pp. 373-390, 2002.

[65] Watanabe Kajiro, Matsuura Ichiro, Abe Masahiro, Kubota Makoto, and Himmelblau D. M., "Incipient Fault Diagnosis of Chemical Processes via Artificial Neural Networks," *AIChE Journal*, vol. 35, no. 11, pp. 1803-1810, 1989.

[66] Levenberg Kenneth, "A Method for the Solution of Certain Non-linear problems in Least Squares," *Quarterly of applied mathematics*, vol. 2, no. 2, pp. 164-169, 1944.

[67] Marquardt Donald W., "An Algorithm for Least-squares Estimation of Nonlinear Parameters," *Journal of the Society for Industrial and Applied mathematics*, vol. 11, no. 2, pp. 431-441, 1963.

[68] Wilamowski and Yu Hao, "LevenbergMarquardt Training (chapter 12)," in *Intelligent Systems*, CRC Press, 2011, pp. 12-1 -12-15.

[69] Habibi Saeid, "The Smooth Variable Structure Filter," *Proceedings of the IEEE*, vol. 95, no. 5, pp. 1026-1059, 2007.

[70] Al-Shabi M., Gadsden S.A., and Habibi S.R., "Kalman Filterning Strategies Utilizing the Chattering Effects of the Smooth Variable Structure Filter," *Elsevier: Signal Processing*, vol. 93, pp. 420-431, 2013.

[71] Ahmed Ryan, "Training of Neural Networks Using the Smooth Variable Structure Filter with Application to Fault Detection," McMaster University, Hamilton, Ontario, 2011.

[72] Chang Jinseok, Kim Manshik, and Min Kyoungdoug, "Detection of Misfire and Knock in Spark Ignition Engines by Wavelet Transform of Engine Knock Vibration Signals," *Institute of Physics Publishing Measurement Science and Technology*, vol. 13, pp. 1108-1114, 2002.

[73] Daniels Chao F., Zhu Guoming G., and Winkelman James, "Inaudible Knock and Partial-Burn Detection Using In-Cylinder Ionization Signal," *SAE Technical Paper Series*, vol. 3149, 2003.

[74] Merkisz Bogus, and Piotr Jerzy, "Short-Time Analysis of Combustion Engine Vibroacoustic Signals Through Pattern Recognition Techniques," *SAE International*, vol. 2529, 2005.

[75] Cavina Nicolo, Corti Enrico, Minelli Giorgio, and Serra Gabriele, "Misfire Detection Based on Engine Speed Time-Frequency Analysis," *SAE International: The Engineering Society for Advancing Mobility Land Sea Air and Space*, vol. 0480, 2002.

[76] Arasaratnam Ienkaran, Habibi Saeid, Kelly Christopher, Fountaine Tony J., and Tjong Jimi, "Engine Fault Detection Using Vibration Signal Reconstruction in the Crank-Angle Domain," *SAE International Noise and Vibration Conference and Exhibition*, p. 12, 2011.

[77] Marinek Radek, Klein Lukas, and Marek Pavel, "Novel Signal Gate Solution Suitable for Implementation in Audio and Recording Technologies," 2012.

[78] Haqshenas Seyyed Reza, "Multiresolution-Multivariate analysis of vibration signals; application in fault diagnosis of internal combustion engines.," McMaster University, Hamilton, Ontario, Canada, 2013.

[79] Kiencke U., "Engine Misfire Detection," *Elsevier*, vol. 7, pp. 203-208, 1999.

[80] Williams Jeremy, "An Overview of Misfiring Cylinder Engine Diagnostic Techniques Based on Crankshaft Angular Velocity Measurements," *SAE INTERNATIONAL*, vol. 960039, pp. 31-37, 1996.

[81] Buede and M. Dennis, The Engineering Design of Systems Models and Methods, Hoboken, New Jersey: John Wiley & Sons, Inc., 2009.

[82] Jakoby Bernhard, Scherr Monika, Buskies Matthias, and Eisenschmid Heinz, "An Automotive Engine Oil Viscosity Sensor," *IEEE Sensors Journal*, vol. 3, no. 5, pp. 1-2, 2003.

[83] Dunbabin Matthew, Brosnan Stephen, Roberts Jonathan, and Corke Peter, "Vibration Isolation for Autonomous Helicopter Flight," in *IEEE International Confernce on Robotics & Automation*, New Orleans, LA, April 2004.

[84] B. M. Reddy, "Dust Resistant Airfoil Cooling". Cincinnati, Ohio Patent 6,164,913, 26 December 2000.

[85] M. E. Moore and J. R. Albers, "High Vibration Electrical Connector". United States of America Patent 5,562,477, 8 October 1996.

[86] Surhone Lambert M., Tennoe Mariam T., and Henssonow Susan F., IBM 7700 Data Acquisition System, Beau Bassin, Mauritius: Betascript publishing, 1964.

[87] IBM Inc., "DPD chronology," IBM, [Online]. Available: https://www.ibm.com/ibm/history/exhibits/dpd50/dpd50_chronology.html. [Accessed 25 08 2017].

[88] Enrique Z.L., "Wikipedia," 27 September 2016. [Online]. Available: https://en.wikipedia.org/wiki/File:DigitalDAQv2.pdf. [Accessed 04 September 2017].

[89] Fausti S.A., Erickson D.A., Frey R.H., and Rappaport B.Z., "The Effect of Impulsive Noise Upon Human Hearing Sensitivity (8 to 20KHz)," *Scandinavian Audiology*, vol. 10, no. 1, pp. 21-29, 1981.

[90] Vegte Van de Joyce , "Nyquist Sampling Theorem," in *Fundamental of Digital Signal Processing*, Upper Saddle River, New Jersey, R.R. Donnelley & Sons, 2001, pp. 30-35.

[91] Webster Ramon Pallas-Areny and Jhon G., Sensors and Signal Conditioning, Toronto, Canada: Jhon Wiley & Sons Inc., 2001.

[92] Walden Robert H., "Analog-to-Digital Converter Survey and Analysis," *IEEE*, vol. 17, no. 4, pp. 539-550, 1999.

[93] Parguian Joselito, "Building a simple data acquisition system using the TMS320C31 DSP," Analog Applications Journal, Dallas, Texas, 2001.

[94] Texas Instruments , "TMS320C31, TMS320LC31 DIGITAL SIGNAL PROCESSORS data sheet," Jan. 1999. [Online]. Available: https://www.mouser.com/catalog/specsheets/tms320c31.pdf. [Accessed 05 Sept. 2017].

[95] Ronald S. Jahr and Thomas H. Cowell, "UTILITY USAGE DATA AND EVENT DATA ACQUISITION SYSTEM". Owosso, Michigan , United States Patent 4,504,831, 12 Mar. 1985.

[96] C. Johnny, "ENVIRONMENTAL SENSOR DATA ACQUISITION SYSTEM". Denton, Texas, United States Patent 5,132,968, 21 Jul. 1992.

[97] Q. Wang, G. Barnett, R. M. Greig and Y. Cheng, "SYSTEM AND METHOD FOR PERFORMING REAL TIME DATA ACQUISITION, PROCESS MODELING AND FAULT DETECTION OF WAFER FABRICATION PROCESSES". Travis, Texas, United States Patent 5,859,964 , 12 Jan. 1999.

[98] Miller Matthew, Bao Mi, Akio Kita, and Ume I. Charles, "Development of automated real-time data acquisition system for robotic weld quality monitoring," *Elsevier Science Ltd. Mechatronics*, vol. 12, p. 1259–1269, 2002.

[99] Bryanta Christopher L. and Gandhi Neeraj J., "Real-time data acquisition and control system for the measurement of motor and neural data," *Journal of Neuroscience Methods*, vol. 142, p. 193–200, 2005.

[100] dSPACE, "dSPACE MicroAutoBox," 2008. [Online]. Available: http://www.ceanet.com.au/Portals/0/documents/products/dSPACE/dspace_2008_microautobox_en_pi480.pdf. [Accessed 19 01 2019].

[101] IBM, "IBM PowerPC 750FX RISC Microprocessor DataSheet," 9 Jun. 2003. [Online]. Available: http://datasheets.chipdb.org/IBM/PowerPC/750/PowerPC-750FX.pdf. [Accessed 05 08 2017].

[102] Steve Corrigan, "Introduction to the Controller Area Network (CAN)," Texas Instruments, May 2016. [Online]. Available: http://www.ti.com/lit/an/sloa101b/sloa101b.pdf. [Accessed 03 08 2017].

[103] Texas Instruments , "ww.ti.com," Mar. 2012. [Online]. Available: http://www.ti.com/lit/ug/sprugp2a/sprugp2a.pdf. [Accessed 04 08 2017].

[104] Hackett Eric, "LIN Protocol and Physical Layer Requirements," Texas Instruments, Feb. 2018. [Online]. Available: http://www.ti.com/lit/an/slla383/slla383.pdf. [Accessed 06 08 2017].

[105] Omar Ali, "Setting up MicroAutoBox 1401/1501 for Power and LIN Communication," Mar. 2014. [Online]. Available:

https://www.egr.msu.edu/classes/ece480/capstone/spring14/group04/documents/Omar.pdf. [Accessed 07 08 2017].

[106] "MIcroAutoBox Embedded PC System Overview," Oct. 1999. [Online]. Available: https://www.dspace.com/files/pdf1/MicroAutoBoxOverviewofBoardRevisions_Release2014B.pdf. [Accessed 03 08 2017].

[107] INTERTECHNOLOGY INC., "AC240 Series DataSheet," [Online]. Available: https://intertechnology.com/CTC/pdfs/AC240_Series.pdf.

[108] CTC , "Power Supplies PS01 Datasheet," [Online]. Available: https://www.ctconline.com/fileup/PrdDS2013/7_PRO_power_supplies.pdf.

[109] Texas Instruments Inc., "CC3200MOD DataSheet," 2014. [Online]. Available: http://www.ti.com/lit/ds/swrs166/swrs166.pdf.

[110] Texas Instruments Inc. , "CC3200MOD SimpleLink™ Wi-Fi® and Internet-of-Things Module Solution, a Single-Chip Wireless MCU," [Online]. Available: http://www.ti.com/product/CC3200MOD/samplebuy.

[111] Texas Instruments Inc. , "KeyStone Architecture Universal Asynchronous Receiver/Transmitter (UART)," Nov. 2010. [Online]. Available: http://www.ti.com/lit/ug/sprugp1/sprugp1.pdf. [Accessed 15 08 2017].

[112] Texas Instruments Inc. , "Understanding the I2C Bus," Jun. 2015. [Online]. Available: http://www.ti.com/lit/an/slva704/slva704.pdf. [Accessed 16 08 2017].

[113] MIT University , "Pulse-width Modulation," Oct. 2000. [Online]. Available: http://fab.cba.mit.edu/classes/961.04/topics/pwm.pdf. [Accessed 20 08 2017].

[114] Texas Instruments Inc. , "KeyStone Architecture General Purpose Input/Output (GPIO) User Guide," Nov. 2010. [Online]. Available: http://www.ti.com/lit/ug/sprugv1/sprugv1.pdf. [Accessed 21 08 2017].

[115] TE Connectivity Inc. , "Model 832M1 Accelerometer DataSheet," Dec. 2017. [Online]. Available: https://www.te.com/commerce/DocumentDelivery/DDEController?Action=showdoc&DocId=Data+Sheet%7F832M1_Accelerometer%7FA%7Fpdf%7FEnglish%7FENG_DS_832M1_Accelerometer_A.pdf%7FCAT-EAC0015. [Accessed 02 10 2017].

[116] InvenSense Inc. , "INMP401 Omnidirectional Microphone with Bottom Port and Analog Output Datasheet," 21 May 2014. [Online]. Available: https://www.invensense.com/wp-content/uploads/2015/02/INMP401.pdf. [Accessed 03 10 2017].

[117] XILINX Inc. , "JTAG Programmer Guide," 1999. [Online]. Available: https://www.asc.ohio-state.edu/physics/cms/cfeb/datasheets/jtag.pdf. [Accessed 06 10 2017].

[118] Texas Instruments Inc., "CC3200 SimpleLink™ Wi-Fi® and IoT Solution with MCU LaunchPad Hardware User's Guide," Jan. 2015. [Online]. Available: http://www.ti.com/lit/ug/swru372b/swru372b.pdf. [Accessed 20 10 2017].

[119] MIT Electric Vehicle Team, "A Guide to Understanding Battery Specifications," Dec. 2008. [Online]. Available: http://web.mit.edu/evt/summary_battery_specifications.pdf. [Accessed 22 10 2017].

[120] Hyperion Inc. , "https://www.robotshop.com/media/files/pdf/hyperion-g5-50c-3s-1100mah-lipo-battery-User-Guide.pdf," 2016. [Online]. Available: https://www.robotshop.com/media/files/pdf/hyperion-g5-50c-3s-1100mah-lipo-battery-User-Guide.pdf. [Accessed 23 10 2017].

[121] Wang Zhenshi and Wei Xuezhe, "Design Considerations for Wireless Charging Systems with an Analysis of Batteries," *Energies,* vol. 8, pp. 10664-10683, 2015.

[122] Nizam Basharat, "Inductive Charging Technique," *International Journal of Engineering Trends and Technology (IJETT),* vol. 4, no. 4, pp. 1054-1059, 2013.

[123] UDOO , "UDOO X86 Datasheet," 18 10 2016. [Online]. Available: https://www.electronicsdatasheets.com/datasheet/UDOO%20X86%20Datasheet.pdf.

[124] UDOO, "UDOO X86 Documentation," 7 Jan. 2019. [Online]. Available: https://www.udoo.org/docs-x86/Introduction/Introduction.html.

[125] Hewlett-Packard Company Inc., Intel Corporation Inc., Microsoft Corporation Inc., NEC Corporation Inc., ST-NXP Wireless Inc., Texas Instruments Inc., "Universal Serial Bus 3.0 Specification," 12 Dec. 2008. [Online]. Available: https://www.usb3.com/whitepapers/USB%203%200%20(11132008)-final.pdf.

[126] S.B. and M.B. from UDOO Inc., "UDOO X86 User Manual," 2017. [Online]. Available: http://download.udoo.org/files/UDOO_X86/Doc/UDOO_X86_MANUAL.pdf.

[127] Intel Inc., "Intel® Quark™ SE Microcontroller C1000," Feb, 2017. [Online]. Available: https://www.intel.com/content/dam/www/public/us/en/documents/datasheets/quark-c1000-datasheet.pdf.

[128] Bosch Sensortec, "DataSheet BMI160 Small, Low Power Inertial Measurement Unit," 10 Feb. 2015. [Online]. Available: https://www.mouser.com/datasheet/2/783/BST-BMI160-DS000-07-1366203.pdf.

[129] Carter Brown and Thomas Bruce R., HANDBOOK OF OPERATIONAL AMPLIFIER APPLICATIONS, Dallas, Texas, United States: Texas Instruments Inc., 2001.

[130] Texas Instruments , "LM741 Operational Amplifier," Oct. 2015. [Online]. Available: http://www.ti.com/lit/ds/symlink/lm741.pdf.

[131] Texas Instruments Inc. , "Getting Started with TINA-TI™," Aug. 2007. [Online]. Available: http://www.ti.com/lit/ug/sbou052a/sbou052a.pdf.

[132] Electronics Tutorials, "Passive High Pass Filter," 2014. [Online]. Available: https://www.electronics-tutorials.ws/filter/filter_3.html. [Accessed 03 11 2017].

[133] Electronics Tutorials , "Passive Low Pass Filter," 2014. [Online]. Available: https://www.electronics-tutorials.ws/filter/filter_2.html. [Accessed 03 11 2017].

[134] Childers Jim from Texas Instruments Inc., "ADC Source Impedance," Aug. 2004. [Online]. Available: http://www.ti.com/lit/an/spna088/spna088.pdf.

[135] Adafruit Technologies Inc. , "Electret Microphone Amplifier - MAX4466 with Adjustable Gain," [Online]. Available: https://www.adafruit.com/product/1063.

[136] CUI Inc. , "Electret Condenser Microphone Datasheet," Jun. 2008. [Online]. Available: https://cdn-shop.adafruit.com/datasheets/CMA-4544PF-W.pdf.

[137] MAXIM, "Low-Cost, Micropower, SC70/SOT23-8, Microphone Preamplifiers with Complete Shutdown," Apr. 2001. [Online]. Available: https://cdn-shop.adafruit.com/datasheets/MAX4465-MAX4469.pdf.

[138] Earl Bill, "Adafruit Learning System: Adafruit Microphone Amplifier Breakout," 22 Aug. 2018. [Online]. Available: https://cdn-learn.adafruit.com/downloads/pdf/adafruit-microphone-amplifier-breakout.pdf.

[139] Cockrill Chris from Texas Instruments Inc., "Understanding Schmitt Triggers," Sep. 2011. [Online]. Available: http://www.ti.com/lit/an/scea046/scea046.pdf.

[140] Wens M. and Steyaert M., "Chapter 2 - Basic DC-DC Converter Theory," in *Design and Implementation of Fully-Integrated Inductive DC-DC Converters in Standard CMOS*, Springer Inc., 2011, pp. 27-63.

[141] Burelle Carol, "On-Board Diagnostics II (OBDII) and Light-Duty Vehicle Emission Related Inspection and Maintenance (I/M) Programs," Apr. 2004. [Online]. Available: https://www.ccme.ca/files/Resources/air/mobile_sources/jia_trnsprt_obd_e.pdf.

[142] Roberts Steve, DC/DC BOOK OF KNOWLEDGE: Practical tips for the User, Gmunden, Austria: RECOM Engineering GmbH & Co KG,, 2014.

[143] Wind River Systems Inc. , "Chapter 3: Polling and Interrupt Handling," Tornado Device Driver Workshop, [Online]. Available: http://read.pudn.com/downloads124/ebook/526229/ddPolIntr_3.f.pdf.

[144] Tatham Simon, "PuTTY User Manual," 1977-2017. [Online]. Available: https://documentation.help/PuTTY/documentation.pdf.

[145] Schaefer Andrew and Schwar Rohde from R&S RTO Inc., "The Effective Number of Bits (ENOB) of my R&S Digital Oscilloscope Technical Paper," Apr. 2011. [Online]. Available: https://cdn.rohde-schwarz.com/pws/dl_downloads/dl_application/application_notes/1er03/ENOB_Technical_Paper_1ER03_1e.pdf.

[146] Frenzel Lou from Design Electronic, "What's The Difference Between Bit Rate And Baud Rate?," 27 Apr. 2012. [Online]. Available: https://www.rpi.edu/dept/ecse/mps/Bit_&_Baud_Rate.pdf.

[147] Sparkfun Electronics Inc. , "Teensy 3.6," 07 May 2014. [Online]. Available: https://media.digikey.com/pdf/Data%20Sheets/Sparkfun%20PDFs/DEV-14057_Web.pdf.

[148] (NXP) Freescale Semiconductor Inc. , "Kinetis K66 Sub-Family 180 MHz Arm Cortex-M4F Microcontroller.," 03 Jan. 2016. [Online]. Available: https://www.pjrc.com/teensy/K66P144M180SF5V2.pdf.

[149] Analog Devices Inc. , "LTC 1863/ LTC 1867 12-/16-bit, 8-channel 200Ksps ADCs datasheet," [Online]. Available: https://www.analog.com/media/en/technical-documentation/data-sheets/ltc1863-ltc1867.pdf.

[150] Architektur akomputerów from Informatyka , "Serial Communication Buses," Jan. 2015. [Online]. Available: http://fiona.dmcs.pl/ak/serial-buses_Jan2015.pdf.

[151] Kerrisk Michael and Jambit, "Linux Programmer's Manual," 29 Oct. 2018. [Online]. Available: http://man7.org/linux/man-pages/man3/termios.3.html.

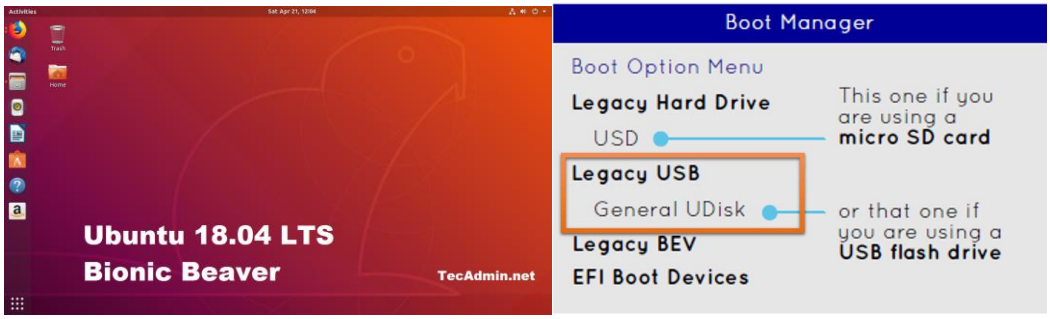
Appendix A.

Operating the FDD System

The FDD system is a general purpose computer that was tailored to carry data acquisition for Fault Detection and Diagnosis. To run the system as a DAQ device, there are two steps to follow: Initialization and Data Collection.

Initialization

1- Install the Linux operating system on the UDOO x86 ULTRA device. In this work, the used version was Ubuntu 18.04.1 LTS. The ISO file should booted on the USB flash drive with a minimum capacity of 8 GB to be able to load the Operating System (OS) on the board (no CD or DVD drive on the UDOO board). Once the USB is connected, the device should be powered while pressing the 'esc' key to reach the boot manager of the BIOS. Once in the boot manager window, select the 'legacy USB' (orange rectangle) in the boot option menu which allows the board to boot from the USB flash drive automatically the first time.

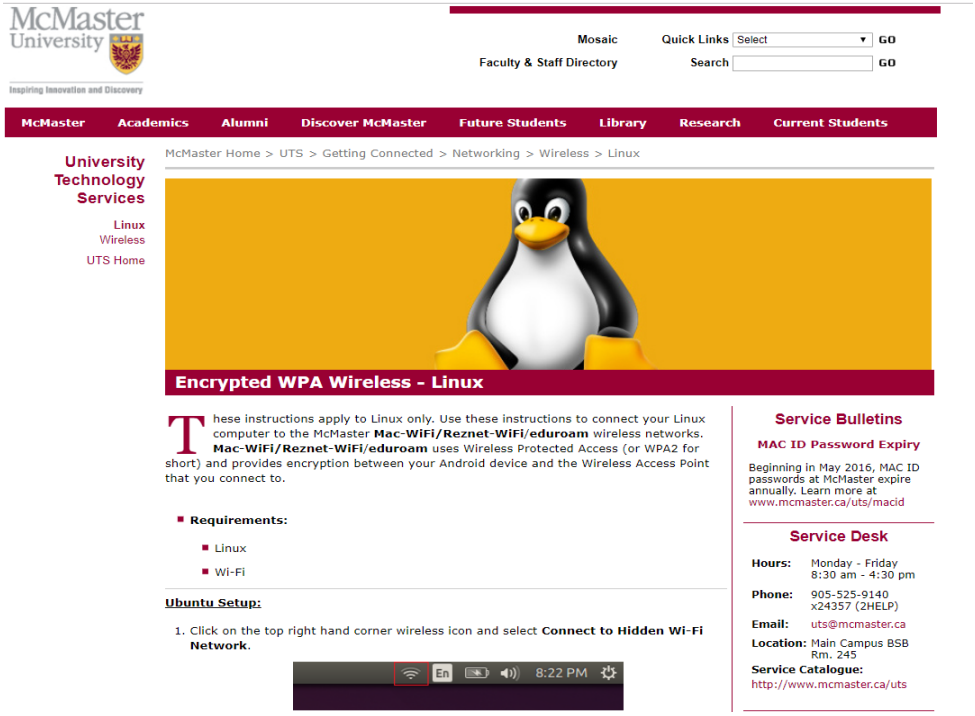


Link 1: https://help.ubuntu.com/community/Installation/FromUSBStick

2- Connect the device to McMaster Wi-Fi. For that, the University Technology Services (UTS) compiled an extensive that can be used as a good reference. You will need to use your McMaster ID and Password for this step. Once the Wi-Fi connection is established, you will need to update the operating system to ensure compatibility with the software packages needed for the FDD application. Updating the system is done through the commands :

Command 1: $ *sudo apt-get update -y*

Command 2: $ *sudo apt-get upgrade -y*



Link 2: https://www.mcmaster.ca/uts/network/wireless/linux_settings.htm

3- Install the Arduino IDE on the operating system. This software package is responsible for booting the firmware of the microcontroller (teensy 3.6). After finishing the installation, is it important to check for errors relating serial port (e.g. *Error opening serial port…*) which

requires setting the serial port permission as depicted by the link 3. To boot the firmware part that relate to the Braswell microprocessor, it is mandatory to compile and execute .c files on the operating system. For that, install the GCC package using command 1 and verify the installation using command 2:

Command 1: *$ sudo apt install gcc*

Command 2: *$ gcc --version*

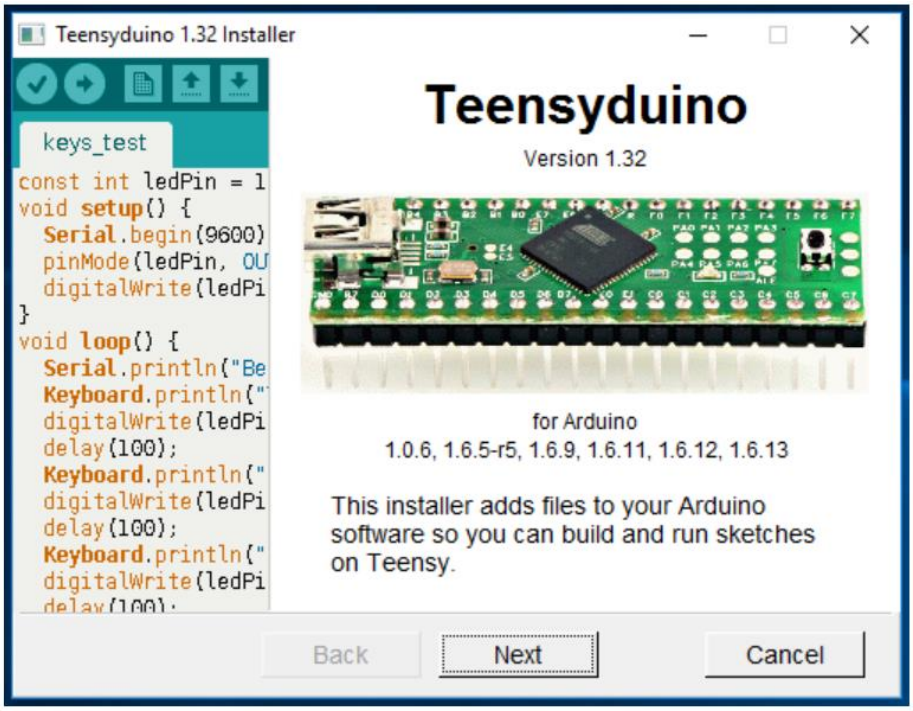
Link 3: https://www.arduino.cc/en/guide/linux

Link4: https://linuxconfig.org/how-to-install-gcc-the-c-compiler-on-ubuntu-18-04-bionic-beaver-linux

4- Install the Teensyduino on the Arduino IDE. For that, Linux Installer (X86 64 bit) should be selected among the available options provided by link 5. When the installer is running, you will be prompted to choose the location for the installation. This location should be redirected to Arduino IDE's file location. To allow non-root users, such as serial peripherals, to have access to the teensy board, the rules depicted in link 6 should be saved to a file named : 49-teensy.rules (.rules is the extension of the file). Later on, this file should

be copied to the following directory: /etc/udev/rules.d/. This is carried out using the following terminal command.

Command : sudo cp 49-teensy.rules /etc/udev/rules.d/



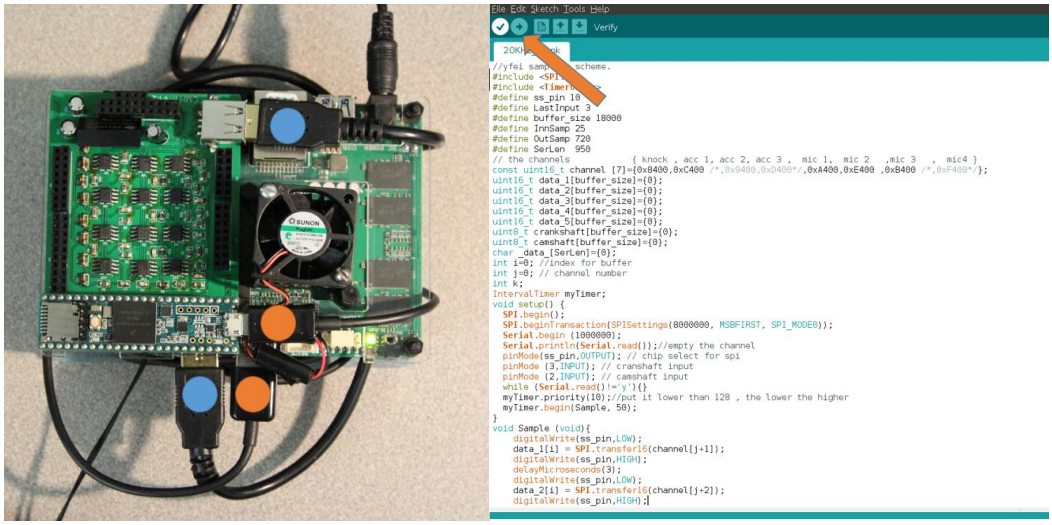
Link 5: https://www.pjrc.com/teensy/td_download.html

Link 6: https://www.pjrc.com/teensy/49-teensy.rules

5- Install Matlab software package along with the following toolboxes: Wavelet Toolbox, Statistics and Machine Learning Toolbox, Signal Processing Toolbox, DSP System Toolbox, and Deep Learning Toolbox. The signal processing codes were tested for two version of Matlab: R2017a and R2018b. Older versions of Matlab, such as R2010a, might have some problems using the aforementioned toolboxes.

Data Collection

1- Power the system, and establish the required USB connections as indicated by the figure on the left. Once this is done, you need to upload the code to the Teensy using Arduino IDE. After selecting the sampling scheme (20 kHz, 40 kHz, and 125 kHz) and making the right adjustments to the code, you need to click on the upload button as depicted by the figure on the right. This compiles the code and uploads it to the teensy board. Once this is done, the teensy executes the setup section of the code and remains in 'waiting mode' until the second firmware is booted to the microprocessor to carry out the communication with it.



2- Before compiling and executing the C codes for the microprocessor, you need to make sure that the serial peripheral number matches the teensy board's (ttyACM0 or ttyACM1). Furthermore, the data path used to save the sensory information in .txt file should match the one used by Matlab which reads the same file later on.

```
void main(int argc, char **argv)
{
        //----------------------- Variables Initialization -------------------//
        if (argc > 1){
                printf("waiting for 40 secs....\n");
                sleep (40);
        }
        int fd;                         //the serial port (USB)
        char read_buffer[950];          //read buffer from the serial port
        char garbage_buffer[950];       //read the garbage data
        int i = 0;                      //counts the number of experiments
        struct timeval tv,ed,ptv,ped;           //time logging variables
        int exp_num = 15;               //save the number of the experiment
        int exp_typ = 99;               //save the experiment type for the fault
        int n;                          //returns the number of reads
        char numconv [13];              //intermediate variable for conversion
        char file_name [13];            //name of the file
        char path [52] ="/home/ahmed/Desktop/PCA_FDD/data_20KHZ/";
        FILE *f;
        id_t pid;
        int ret;
        //----------------------- Serial Communication Set-up ----------------//
        gettimeofday(&ptv,NULL);
        //printf("prior is %d \n", getpriority(PRIO_PROCESS,getpid()));
        //ret = setpriority(PRIO_PROCESS, getpid(), -20);
        nice(-20);
        //printf("posterior is %d \n", getpriority(PRIO_PROCESS,getpid()));
        //printf("return valus %d \n", ret);
        //fd = open("/dev/ttyACM1",O_RDWR | O_NONBLOCK);
        fd = open("/dev/ttyACM0",O_RDWR | O_NOCTTY | O_NDELAY);
        if(fd == -1)
           printf("\n Error! in Opening USB port  \n ");
        else
           printf("USB port Opened Successfully \n");
        struct termios SerialPortSettings;
        tcgetattr(fd, &SerialPortSettings);
        cfsetispeed(&SerialPortSettings,B38400); //read speed
        cfsetospeed(&SerialPortSettings,B38400); //write speed
        // 8N1
        SerialPortSettings.c_cflag &= ~PARENB;
        SerialPortSettings.c_cflag &= ~CSTOPB;
```

Data file path

USB peripheral

3- Run and compile the C codes on Linux kernel. The appropriate code should be selected to either do testing or data collection. To carry a test, only one data file is saved to the pre-set path. To carry data collection, the FDD system logs a pre-set number of data files into the saving directory. To compile the code, command 1 is used where ‘collect_40KHZ.c’ is the name of the .c code. Meanwhile, to execute the code and start the logging process, command 2 is typed on the terminal which boots the code on the kernel with the highest priority. To further carry either testing or data collection, the second command is repeated accordingly.

Command 1: gcc collect_40KHZ.c –o collect

Command 2: sudo nice –n -20 ./collect

4- Once the text file is saved in the appropriate directory, the applicable Matlab code (depending on the FDD technique) is executed. This code opens the text file under its working directory and runs the appropriate signal processing technique.

Appendix B.

The Pseudocode of the Teensy's Firmware C/C++

Note: Every text indicated by '//' is a comment on the pseudocode

//Teensy 3.6 Sampling Code

//Section 1: Initialization Process

INCLUDE the required libraries 'SPI' and 'TimerOne'

SET the SPI chip select variable 'ss_pin' to be pin#10 of the Teensy board

SET the data transfer 'buffer_size' variable to 15000

SET the 'InnSamp' variable to 25

SET the 'OutSamp' variable to 600

SET the 'SerLen' variable to 950

INITIALIZE the ADC channels to be sampled in the 'channel[7]' array variable

INITIALIZE 6 data buffers' content (data_1[buffer_size] .. data_6[buffer_size]) to 0

INITIALIZE the content of the 'camshaft[buffer_size]' to 0

INITIALIZE the global buffer '_data_[SerLen]' to 0

INITIALIZE all the loop indices i,j, and k to 0

CREATE a new timer interrupt named 'myTimer'

// Section 2: the setup function of the Code

Function setup () {

    //Used to setup the FDD environment

    CALL SPI.begin() function with no arguments to initialize the SPI communication

    CALL SPI.beginTransaction() function with arguments: 8000000, MSBFIRST, and SPI_MODE0 to set the SPI baud at 8000000, the bit order at MSBFIRST and the SPI mode at mode 0

    CALL Serial.begin() function with argument equal to 1000000 to Initialize the Serial Communication's baud rate

CALL Serial.println() function with argument Serial.read() function to Empty the serial buffer from garbage bits

CALL pinMode() function with arguments: 'ss_pin' and OUTPUT to set SPI Chip select 'ss_pin' as OUTPUT

CALL Serial.read() function with no arguments to read the serial channel

WHILE the output of Serial.read() is not equal to 'y'

CALL Serial.read() function with no arguments

CALL myTimer.priority() function with input argument 10 to set the timer priority to 10 (1 being the highest priority)

CALL myTimer.begin() function with two input arguments: Sample() function and 50 to initialize the timer interrupt callback function Sample () with sampling interval of 50 microseconds.

}

//Section 3: All defined functions in the code

Function sample (){

//The sampling callback function to the interrupt

FOR channel 'i' in all sensory channels

CALL digitalWrite () function with input arguments: ss_pin and LOW to set the chip select pin 'ss_pin' to LOW which enables the ADC module.

CALL SPI.transfer16() with the i-th channel's 2-bytes HEX address as argument

SAVE the incoming 2 bytes from the function into the 'i'th data buffer

SET the chip select pin 'ss_pin' to HIGH which disable the ADC module

DELAY 3 microseconds

CALL digitalRead() function with argument 3 which represent the camshaft pin address

SAVE the output of digitalRead() in the camshaft[] buffer array.

}

// Section 4: the loop function of the code

Function loop () {

// the main function

IF the variable ‘i’ is bigger or equal the buffer_size

CALL myTimer.end() function with no arguments to stop the interrupt routine

DELAY 100 microseconds

SET k equal to 0

WHILE k less than OutSamp

ADD 1 to k

CALL Serial.read() function with no arguments

WHILE the output of Serial.read() is not equal to 'a'

CALL Serial.read() function with no arguments

SET i equal to 0

WHILE i less than InnSamp

ADD 1 to i

FILL the main buffer _data_ with samples from all 6 data buffers

FILL the main buffer _data_ with samples from the camshaft buffer

ADD the end of line character '\n' to the _data_ buffer

CALL Serial.write () function with 2 arguments: _data_ and SerLen

CALL Serial.read() function with no arguments

WHILE the output of Serial.read() is not equal to 'y'

CALL Serial.read() function with no arguments

SET i to 0

CALL myTimer.begin() function with two input arguments: Sample() function and 50 to initialize the timer interrupt callback function Sample () with sampling interval of 50 microseconds.

}

Appendix C.

The Pseudocode of the Microprocessor Firmware C/C++

// Pseudo code for the Braswell Microprocessor

// SECTION 1: Load the required libraries

INCLUDE the required libraries 'stdio', 'stdlib', 'time', 'fcntl', 'termios', 'unistd', 'errno', 'sys/time', 'string', 'sys/resource'

// SECTION 2: Define the used functions

Function delay(number_of_seconds)**{**

//function that adds delay to the operations

INITIALIZE 'milli_seconds' to be equal to 1000 times the variable 'number_of_seconds'

CALL the function clock() with no input arguments

INITIALIZE the variable 'start_time' using the function clock() which is called with no argument

WHILE the result of clock() function is less than the sum of 'start_time' and 'milli_seconds' variables

CALL the function clock() with no input arguments

}

// SECTION 3: The main function of the code

Function main (argc, argv){

// PART 1: Define the variables

IF argc is strictly bigger than 1

PRINT the message "waiting for 40 secs ..... \n"

CALL the function sleep() with the input argument 40

INITIALIZE the variable 'fd' which is used to open the serial peripheral

INITIALIZE the 'read_buffer[950]' char array to save the incoming messages from teensy

INITIALIZE the 'garbage_buffer[950]' char array to read any garbage data in the buffer

INITIALIZE the variable 'i' to 0 to save the number of loggings that are carried

INITIALIZE the variables 'tv','ed','ptv', and 'ped' to save the time elapsed of different functions within the main() function

INITIALIZE the variable 'exp_num' to 15 which saves the maximum number of files to be sampled during data collection

INITIALIZE the variable 'exp_typ' to the fault category that is being induced on the engine during data collection

INITIALIZE the variable 'n' that saves the number of reads that are carried out by the microprocessor

INITIALIZE the 'numconv[13]' char array that acts as an intermediate step to generate the file names

INITIALIZE the 'file_name[13]' char array that saves the name of the text files that are being logged

INITIALIZE the 'path[52]' char array to "path/of/the/collected/data"

INITIALIZE the pointer variable 'f' to objects of type FILE which saves the return of fopen() function

INITIALIZE the 'pid' variable that saves the output of the function getpid() which returns the current process id that is running on linux kernel

INITIALIZE the 'ret' variable to save the return of the setpriority() function

CALL the function gettimeofday() with 2 agruments: &ptv and NULL to log the current time of the execution into ‘ptv’ variable

CALL the nice() function with argument equal to -20 to set the main thread with the highest priority in the Linux Kernel

// PART 2 : Launch the serial Communication and set its parameters

CALL the open() function with the following arguments: "/dev/ttyACM0" and O_RDWR | O_NOCTTY | O_NDELAY. The first argument is the peripheral path. The second argument is the opening options of the peripheral

SAVE the result of the open() function into the variable fd

IF fd is equal to -1

    PRINT the message "\n Error! in Opening USB port \n"

ELSE

    PRINT the message "USB port Opened Successfully \n"

INITIALIZE the 'SerialPortSettings' termios structure

CALL the function tcgetattr() with the following arguments: fd and &SerialPortSettings (& reference operator) to get the current attributes of the serial port

CALL the functions cfispeed() and cfospeed () with the two arguments: &SerialPortSettings and B38400 to set respectively the read and write speeds to 38400 Baud

INITIALIZE the bits in 'SerialPortSettings.c_cflag' settings to ~PARENB, ~CSTOPB,~CSIZE, CS8,CREAD | CLOCAL, and ~CRTSCTS which correspond to 8 data bits, no parity bit, 1 stop bit, and no flow control serial communication option (8N1) which matches data packets received from the Teensy

CALL the function fcntl() with the following arguments: STDIN_FILENO,F_SETFL, and O_NONBLOCK which makes the reads of the Teensy's data packets non blocking

INITIALIZE the bits in 'SerialPortSettings.c_iflag' settings to ~(IXON | IXOFF | IXANY) which turns off all software flow control of the serial communication

INITIALIZE the bits in 'SerialPortSettings.c_lflag' and 'SerialPortSettings.c_oflag' settings to ~(ICANON | ECHO | ECHOE | ISIG) and ~OPOST respectively to collect the raw packets from the serial communication

SET the element VMIN in the array 'SerialPortSettings.c_cc[VMIN]' to 12 which reflects the number of chars that determine the completion of a read functions

SET the element VTIME in the array 'SerialPortSettings.c_cc[VTIME]' to 0 which reflects the time it takes between two consecutive byte transfers

CALL the tcsetattr() function with the following arguments: fd, TCSANOW, and &SerialPortSettings to set the serial port with all aforementioned configuration options

IF the return value of the tcsetattr() function is strictly less than 0

> **CALL** the perror() function with "init_serialport: Couldn't set term attributes" argument

//PART 3: The Data transfer between Teensy and the Braswell

SET the index 'i' to 1

WHILE 'i' is less or equal to the variable 'exp_num'

> **ADD** 1 to 'i'
>
> **CALL** strcpy() function with the arguments path and "path/of/the/collected/data" to save the data path into the variable 'path'
>
> **CALL** strcpy() function with the arguments file_name and "data" to save the string "data" into file_name

CALL sprintf() function with the arguments numconv and "%02d",exp_typ to save the 'exp_typ' as a 2 digit number at the intermediate file name variable 'numconv'

CALL strcat() function with the arguments file_name and numconv to concatenate the string 'numconv' to the string 'file_name'

CALL sprintf() function with the arguments numconv,"%02d", and i to save the current file number 'i' as a 2 digit number at the intermediate file name variable 'numconv'

CALL strcat() function with the arguments file_name and numconv to concatenate the string 'numconv' to the string 'file_name'

CALL strcat() function with the arguments file_name and ".txt" to concatenate the string ".txt" to the string 'file_name'

CALL strcat() function with the arguments path and file_name to concatenate the string 'file_name' to the string 'path'

CALL fopen() function with the arguments path and "w" to open the text file indicated by the path in writing mode

SAVE the return value of fopen() in the variable 'f'

IF f is equal to NULL

    PRINT the message "Error opening file!\n"

    CALL exit() function with argument 1 to stop the program from running

CALL tcflush () function with the arguments fd and TCIFLUSH to flush the serial buffer from garbage bits

INITIALIZE the buffer 'write_buffer[]' to "y"

CALL write() function with the arguments fd, write_buffers, and sizeof (write_buffers) to send the character "y" to teensy board which initializes the sampling of the sensors by the microcontroller

CALL usleep() function with 800000 acting as argument. 800000 is in usec (0.8 seconds) which is the required time by teensy to run the setup() function

INITIALIZE the buffer 'write_buffer[]' to "a"

CALL the function gettimeofday() with 2 arguments: &tv and NULL to log the current time of the execution

SET the index 'k' equal to 0

WHILE the index 'k' is strictly less than 600

ADD 1 to k

CALL write() function with the arguments fd, write_buffers, and sizeof (write_buffers) to send the character "a" to teensy board which signals to the teensy that the microprocessor is ready to read the next data batch in the communicating buffer

CALL usleep() function with 10000 acting as argument. This guarantees enough time for the teensy to fill the buffer with the next data batch

CALL read() function with the arguments fd, &read_buffer, and 950 which reads the common buffer between the microcontroller and the microprocessor

SAVE the return value of read() function in the variable 'n'

IF 'n' is strictly smaller than 896

PRINT the values of 'n' and 'k' for troubleshooting purposes

DECREMENT the index 'i' by 1

ELSE

CALL fprintf() function with the arguments f,"%950s", and read_buffer to save the results of read_buffer into the accessed text file represented by 'f'

CALL the function gettimeofday() with 2 arguments: &ed and NULL to log the current time of the execution

PRINT the value of the time elapsed in seconds to log one data text file

CALL fclose() function with the argument f to close the current text file

CALL fclose() function with the argument fd to close the current serial peripheral

CALL the function gettimeofday() with 2 arguments: &ped and NULL to log the current time of the execution

PRINT the value of the time elapsed in seconds to run the whole program

}