ADC Dynamic Performance and Python-Based Processing Platform for EMG Wearable Sensors

by Marc Rosenthal

A Thesis

Submitted to the Faculty

of the

WORCESTER POLYTECHNIC INSTITUTE

In partial fulfillment of the requirements for the

Degree of Master of Science

in

Electrical and Computer Engineering

May 2024

THESIS COMMITTEE:

Prof. Edward A. Clancy, Research Advisor

Prof. Xinming Huang, Committee Member

Prof. Ziming Zhang, Committee Member

Acknowledgements

I am deeply grateful for several individuals who contributed and guided me over the course of this thesis. Firstly, my thesis advisor, Professor Edward A. Clancy, for his support, encouragement, and guidance in both this research and academics. I would also like to thank Dr. Rabeeh Majidi, CEO of OrthoKinetic Track, who sponsored the work discussed in chapter five of this thesis. Finally, I would like to thank Kiki Rajotte who provided technical guidance to get me started in my research.

Abstract

This thesis describes investigation of the dynamic performance of an analog-to-digital converter (ADC) for wireless wearable systems and separately the development of a Python-based application for electromyography (EMG) data acquisition and processing. To assist in the investigation of ADC dynamic performance, a MATLAB implementation of a time-domain sinusoid fitting technique was developed for effective number of bits (ENOB) estimation. Performance of the ADC was measured against power consumption for various resolution and oversampling configurations on Nordic Semiconductor's nRF52840 system-on-a-chip (SoC). Results showed oversampling to provide a significant improvement in ENOB, though at a substantial relative cost in power consumption. The Python-based application for EMG was developed to facilitate the acquisition, processing, and visualization of biometric data with a user-friendly graphic user interface. The application was developed with a modular design, utilizing object-oriented coding and open-source libraries, to ease future development.

Table of Contents

1 Introduction

Wireless and wearable physiological sensors have become increasingly common in applications of human studies, medical research, patient monitoring, and human-machine interface control. Whether used for remote patient monitoring, sports performance analysis, or rehabilitation, these sensors play a crucial role in capturing and analyzing physiological signals such as electromyography (EMG). The accuracy and efficiency of data acquisition are paramount, as they directly impact the quality of information obtained from physiological measurements.

Electromyography serves as a measure of the electrical activity associated with the activation and contraction of motor units within skeletal muscles. Due to the nature of surface EMG as an amplitude-modulated random process, significant processing must be done to extract useful information from raw EMG. Traditional processing of surface EMG involves several steps, including filtering to eliminate noise, whitening to reduce signal variability, rectification or squaring to estimate amplitude, and smoothing to capture signal trends. Additionally, decimation and noise correction techniques may be applied to improve signal fidelity.

At the core of these sensor systems lies the analog-to-digital converter (ADC), crucial for transforming continuous analog physiological signals into digital data for computational analysis. Key specifications, including resolution, conversion speed, and dynamic performance metrics such as signal-to-noise ratio (SNR), signal-to-noise and distortion ratio (SINAD), and effective number of bits (ENOB), characterize ADC performance. The ENOB metric reflects the usable resolution of the ADC in real-world conditions, considering noise and distortion.

The Nordic Semiconductor nRF52840 wireless microcontroller offers advanced features for internet-of-things (IoT) and wearable applications, including an adjustable 8-channel, successive-approximation register (SAR), 12-bit ADC: low power consumption, and support for multiple wireless protocols. It includes protocols for Bluetooth Low Energy (BLE) communication, which is ideal for physiological monitoring applications, offering low power consumption and high transmission rates.

Chapters 3 and 4 of this thesis focuses on the objectives related to the performance of the nRF52840 for use in the collection of EMG:

1. Effective estimation of ENOB from input test signals.
2. Measurement of ENOB under different available ADC configurations.
3. Measurement of power consumption under the same configurations.
4. Evaluating the trade-off between improvements to signal fidelity and power consumption.

These objectives contribute to understanding the optimization of EMG sensor systems. Understanding the nuances of ADC performance can assist in selection of the right ADC architecture depending on the application.

Chapter 5 of this thesis aims to address the practical implementation of physiological data acquisition, processing, and visualization using a Python-based application. The development of this application is tailored towards facilitating remote patient monitoring and assessment of rehabilitation progress, particularly focusing on shoulder range of motion (ROM) and electromyography (EMG) data.

The application leverages the versatility of Python and integrates various open-source libraries such as Tkinter for the graphical user interface (GUI), Matplotlib for data visualization, NumPy for numerical computing, and SciPy for signal processing. This combination allows for the creation of a modularly designed, user-friendly interface that enables efficient data handling and analysis.

The main program orchestrates the flow of data between different modules, managing tasks such as serial data collection, filtering options, data processing, visualization, and analysis. Users are provided with flexibility in configuring serial ports, defining data packets, and applying filters specific to EMG processing. Real-time plotting and analysis capabilities empower users to monitor joint angles, detect movement cycles, and normalize data for advanced analysis purposes.

Despite its promising functionality, the application acknowledges areas for improvement. Optimization of data handling and filtering algorithms is identified as crucial for enhancing efficiency and responsiveness. Standardization of data export formats will facilitate interoperability with other tools and systems. Additionally, refinement of algorithms will facilitate accurate data analysis calculations. Overall, the application will serve as a useful tool for clinical research, development, and as a proof of concept for a future user-facing application.

2 Background

2.1 Electromyography

Electromyography (EMG) is a measure of the electrical activity associated with the activation and contraction of motor units of skeletal muscles. A motor unit (MU) is defined as a motor neuron and the muscle fibers innervated by that neuron [1]. Human muscles can be categorized into two types: smaller, slow twitch, type I MUs or larger, fast twitch, type II MUs. During muscle contraction, MUs are activated by an action potential, initiated by the motor neuron. The action potential moves along the length of the muscle fiber, producing a motor unit action potential (MUAP). A characteristic shape of a single MUAP signal is shown in Figure 2-1.

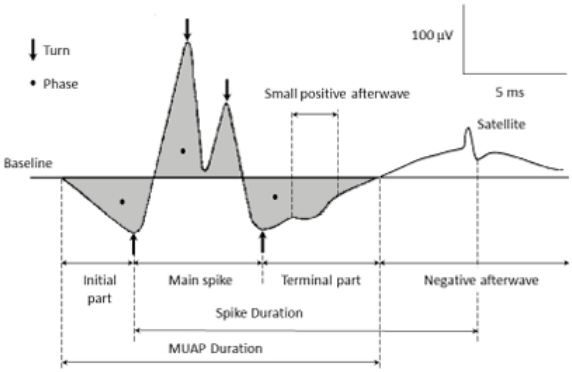

Figure 2-1: Shape of a MUAP signal recorded from an indwelling electrode [2].

To generate a sustained contraction, the motor neuron will repeatedly generate MUAPs at a rate related to the intensity of contraction. As the intensity of muscle contraction is increased, the force generated by a muscle will be increased by recruiting additional MUs and/or increasing the rate at which already active MUs are firing. The temporal sum of MUAPs is measured by electrodes as EMG. A diagram of the generation of the EMG signal from the trains of multiple MUAPs is shown in Figure 2-2.

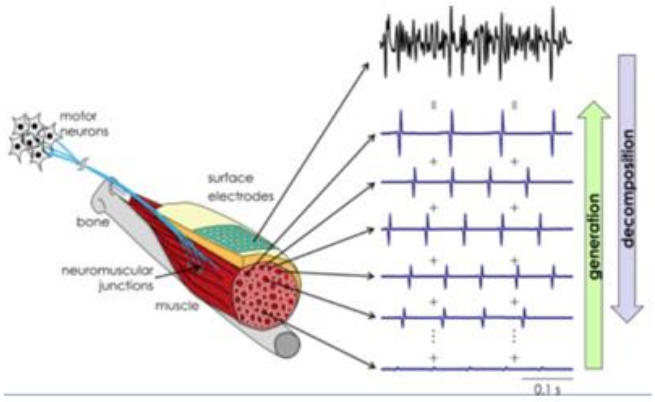

Figure 2-2: Generation and Decomposition of the surface EMG signal [3].

There are two types of electrodes commonly used for EMG collection — surface and intramuscular, with surface being the most common due to being non-invasive. Surface electrodes collect the contributions of MUAPS in a measure inversely proportional to the distance between the motor unit and the electrode on the skin. Due to this, the magnitude of surface EMG (sEMG) is not a direct measure of the magnitude at the motor neuron [4].

2.1.1 Characteristics of EMG Signals

Surface EMG signals are generally modeled as amplitude modulated stochastic (random) processes that can be characterized by a probability density function, often Gaussian or Laplacian. Typical measured surface electrode amplitudes range from 0 to 10 mV (peak-to-peak) or 0 to 1.5 mV (rms) with frequencies within the range 0 to 1000 Hz, with the dominant signal power being within 0 to 150 Hz. Due to the contributions of noise in the signal, the band of usable signal is generally limited to 0 to 500 Hz. There are various common sources of noise included within raw measured EMG including: noise generated from collection equipment, ambient power line interference, motion artifacts, electrode-skin interfacing, and random firings of motor units [5].

2.1.2 Traditional Surface EMG Processing for Amplitude Estimation

As EMG is the measure of MUAPs which have both negative and positive components, whereas the induced MU force generation is only positive, making EMG representative of contraction intensity (force) processing must be done. Figure 2-3 outlines the processing steps for amplitude estimation from sEMG [6].

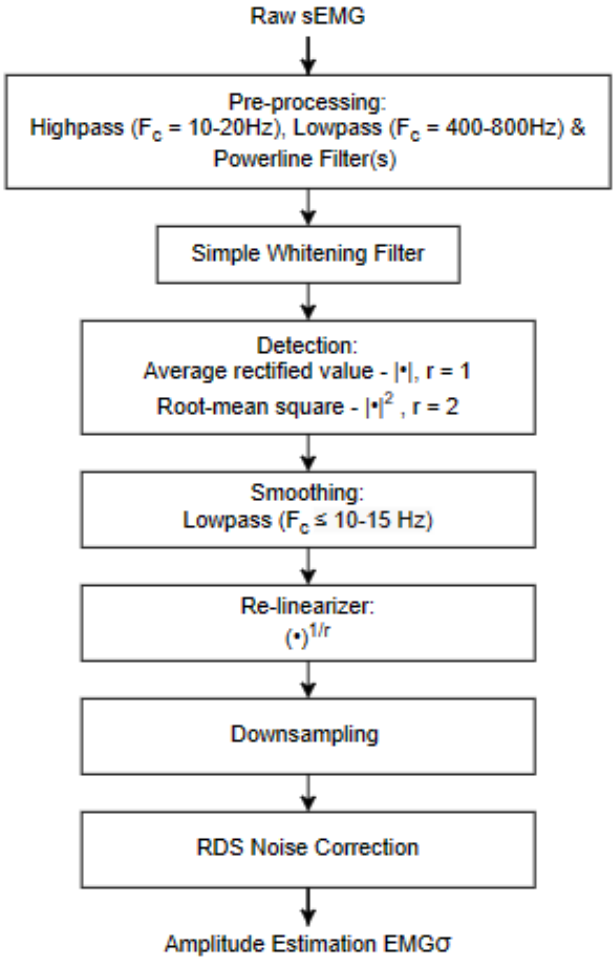

Figure 2-3: Filter steps used to produce EMG amplitude estimation, adapted from [6]

The goal of pre-processing is to eliminate outside noise acquired along with the sEMG signal. A highpass filter with a cutoff frequency of 10-20 Hz can be used to eliminate DC offsets and reduce motion artifacts and other unwanted instability in the signal. The filter order of 2 or 4 is common for the highpass but can be as high as 6 or 8 in applications with large motion artifacts. Since the band of usable signal exists below 500 Hz, a lowpass filter of order 2 or 4 can be applied with a cutoff of 400-800 Hz. Conventional filter types for the low-pass and highpass filters are Butterworth or Chebyshev. A notch filter can be used to eliminate power-line interference at 50/60Hz, with additional cascaded notch filters at harmonics that fall within the targeted signal band. Since power-line interference is extremely narrow, the filter should be designed with a width of 1 Hz, to limit impact on other relevant frequencies. An optional whitening filter can be used to statistically uncorrelate the EMG and reduce the variance of the signal. A simplified whitening filter (e.g. a first difference) can be used for this step. The next step is to estimate sEMG amplitude as a

function of time, by “detecting” the voltage modulation, which is representative of contraction intensity. To achieve this, one of two “detection” methods can be used, either an absolute value (rectification) or squaring operation. The signal then should be smoothed to acquire the signal trend of the sEMG amplitude (low frequency components). To do this, either a moving average or conventional low-pass filter can be used. For low-pass filtering, filter orders of 2 or 4 with cutoff frequencies of less than 10-15 Hz are common. If a squaring operation is used during the detection step, the signal should be re-linearized (square root operation) following the smoothing low-pass filter. An optional decimation step can be applied following smoothing. The decimation operation should include a cascade of a lowpass filter to attenuate the signal above the Nyquist frequency (half of the resultant sampling rate) to avoid aliasing. The smoothing lowpass filter from the prior step may be used to provide this anti-aliasing.

If an estimate of the standard deviation of noise is available, which is traditionally computed from a sample of sEMG at rest, a further step of root difference of squares (RDS) noise correction can be applied. The goal of this step would be to attenuate (broad band) background noise which cannot be removed with the steps from preprocessing.

2.2 Wireless Sensor Systems

Wireless and wearable physiological sensors have become common for the applications of human studies, medical research, patient monitoring, and human-machine interface control [7, 8]. Most applications of wearable physiological sensors require collection over multiple channels, potentially across multiple peripheral nodes, to be transferred to a central node for further processing. Typically, each peripheral node includes its own analog-to-digital converter (ADC), to sample connected analog channels before packaging and broadcasting to the central node [8].

2.2.1 Bluetooth Low Energy

A commonly used communication protocol for biomedical signal applications is Bluetooth Low Energy (BLE) for its low power consumption, high transmission rate, and low cost. BLE provides the flexibility to adjust parameters such as connection interval, maximum transmission unit (MTU) size, and event length, to best suit the needs of the applications and the number of peripheral nodes being supported [7]. The connection interval controls the time interval between successive peripheral-central device connection events. The MTU size is the maximum packet size that can be sent over one transmission. Event length is the time interval reserved for sending and receiving packets. Understanding the effect these parameters have on power consumption and transmission integrity is important for designing a wearable sensor system.

2.2.2 Analog to Digital Converter

ADCs are devices which translate analog real-world signals into digital representations which can be interpreted by digital processors for further manipulation,

transmission, or storage. The first stage of an ADC is typically a sample and hold circuit which stores a value of the continuous analog signal at a defined sampling rate. This signal is digitized by the converter by comparing the "held" analog value to reference voltages of different quantization levels. Parallel converters, such as found in flash ADCs, simultaneously compare the quantization levels. Serial converters, such as found in successive approximation register (SAR) ADCs, compare each quantization level in succession [9].

There are several key specifications beyond converter type that can be used to characterize an ADC. The DC specifications define the conversion range, in which input voltages can fall without being clipped or causing distortion. This range is typically defined by the ADC's power supply voltages. Resolution, measured in bits, indicates the number of quantization levels the analog sample will be compared to, as well as the size of the resultant digital output. Conversion speed determines the fastest sampling rate capability of the ADC [10].

There are several common metrics used to evaluate the dynamic performance of an ADC such as signal-to-noise ratio (SNR), total harmonic distortion (THD), total harmonic distortion plus noise (THD + N), signal-to-noise and distortion ratio (SINAD), effective number of bits (ENOB), and spurious free dynamic range (SFDR) [11]. Most methods for quantifying these metrics involve analysis in the frequency domain, which can be done by estimating the power spectral density (PSD) of the output of ADC.

2.2.2.1 Signal-to-Noise Ratio

Signal-to-noise ratio (SNR) is defined as the ratio of the power of the wanted components of the signal to the power of broadband background noise or otherwise unwanted components. Note that distortion (harmonic, intermodulation, etc.) is *not* included in the calculation of broadband background noise. You can express SNR in terms of decibels using the equation:

$$SNR_{dB} = 10 * log_{10}\left(\frac{P_{signal}}{P_{noise}}\right)$$

In an ideal ADC, the only noise would be error due to the quantization of the signal. Quantization error is due to there being a range of analog values that will be interpreted as a discrete digital value, where the range is the smallest voltage change represented by a bit, or least-significant bit (LSB). A simplification of this error is shown in Figure 2-4. The size, in volts, of the LSB is determined by the voltage reference and the resolution (N-bits) of the ADC [12].

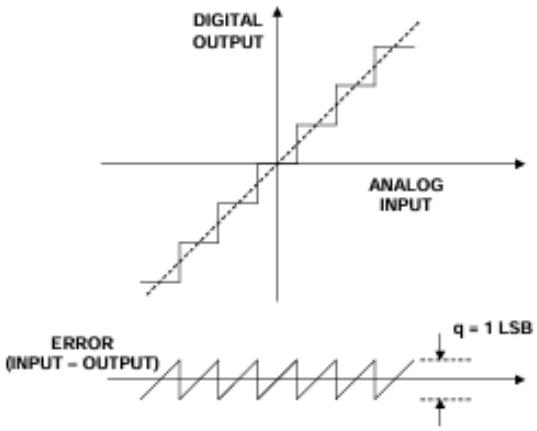

Figure 2-4. Transfer Function and Quantization Noise of an Ideal N-bit ADC [12].

Using this simplified model, the SNR of an ideal N-bit ADC can be estimated by the equation:

$$SNR_{dB,ideal} = (6.02 * N + 1.76)\ dB$$

This ideal SNR can then be compared to the actual SNR of the ADC to evaluate the performance of the ADC, where a higher SNR indicates a higher quality system.

2.2.2.2 Signal-to-Noise and Distortion Ratio

Signal-to-Noise and Distortion Ratio (SINAD) is a crucial metric used to evaluate the dynamic performance of an ADC. Unlike SNR, which solely considers the ratio of the power of the wanted signal to the power of broadband background noise, SINAD factors in both the signal and all unwanted distortions (including, for example, harmonic and intermodulation distortion—which is not included in the noise portion of an SNR measurement). It provides a more comprehensive understanding of the ADC's performance in real-world scenarios where signal integrity can be affected by various sources of noise and distortion.

SINAD is calculated as the ratio of the power of the signal to the sum of the power of all non-signal components, including noise and distortion. You can express SINAD in terms of decibels using the equation:

$$SINAD_{dB} = 10 * log_{10}\left(\frac{P_{signal}}{P_{noise} + P_{distortion}}\right)$$

In an ideal scenario, where there is no distortion present apart from quantization noise, SINAD would essentially represent the SNR. However, in practical ADC implementations, various sources of distortion, such as harmonic distortion and intermodulation distortion, can affect the overall performance. A high SINAD value indicates that the ADC effectively preserves the integrity of the input signal, even in the presence of noise and distortion [11].

2.2.2.3 Effective Number of Bits (ENOB)

Effective number of bits (ENOB) is a measure of the usable resolution in number of bits of the ADC when noise and distortion are considered. It provides an intuitive measure of the dynamic quality of the ADC to be compared to the ADC's stated resolution. ENOB is calculated by comparing the calculated SINAD of the ADC to the theoretical SNR of an ideal N-bit ADC:

$$ENOB = \frac{SINAD_{dB} - 1.76}{6.02}\ bits.$$

It should be noted that this equation assumes that the input signal is a sine wave that fills the full input range of the ADC. However, in real world systems, approaching the full input range may result in clipping or additional distortion due to exceeding the ADC's conversion range. If the magnitude of the input signal is lowered below the full ADC range, the measured value of SINAD using the above equation will be reduced. Thus, a correction factor is typically applied for calculating ENOB:

$$ENOB = \frac{SINAD_{dB} + 20 * log_{10}\left(\frac{V_{ref}}{V_{input}}\right) - 1.76}{6.02}\ bits,$$

where V_{ref} is the full peak-to-peak range of the ADC, and V_{input} is the peak-to-peak range of the input signal.

ENOB essentially indicates the equivalent number of ideal bits that the ADC effectively utilizes under real-world operating conditions. A higher ENOB value signifies better dynamic performance, indicating that the ADC can maintain high accuracy and fidelity in converting analog signals to digital representations, even in the presence of noise and distortion [11].

2.2.3 Nordic NRF52840 Wireless Microcontroller

The nRF52840 microcontroller is an advanced system-on-chip (SOC) designed by Nordic Semiconductor ideal for internet-of-things (IoT) devices, wearables, and other wireless system applications. The nRF52840 supports multiple wireless data transmission protocols, including Bluetooth mesh, Bluetooth Low Energy (BLE), Thread, and ZigBee. The nRF52840 features an adjustable 8-channel successive-approximation register (SAR) ADC which can be configured to convert data at resolutions of 8, 10, 12, or 14 bits (14 bits is achieved with oversampling), at a maximum speed of 200 ksps. With adaptive on-chip power management capabilities it has low power consumption. The nRF52840 can act as both peripheral and central node, with a variety of peripheral input/output (IO) ports and virtual UART serial ports [13].

2.3 Muscle Activity, Human Motion, and Joint Kinematics

Wireless sensor systems enable continuous monitoring of physiological data and can be integrated into wearable devices or attached directly to the skin, allowing for unobtrusive and convenient data collection during rehabilitation sessions. Studies have demonstrated the effectiveness of wireless sensor systems in monitoring and guiding rehabilitation for injuries and conditions [14]. Additionally, wireless sensor systems offer the advantage of remote monitoring, allowing clinicians to assess patients' progress and adherence to prescribed exercises outside of clinical settings. This remote monitoring capability enhances the continuity of care and enables timely adjustments to rehabilitation protocols based on objective data.

2.3.1 Estimation of Muscle Contribution from Electromyography

Electromyography (EMG) data as a measure of motor unit activation is often used to estimate the production of force generation by the measured muscle. However, the nature of measured EMG, and different factors such as the dynamics of muscle tension production, muscle fiber type, fiber length, and joint kinematics complicate the relationship between EMG and force [15]. Due to this, it is common for EMG data across different muscles to be normalized to its respective EMG during maximal voluntary contraction [16]. Additionally, it is generally understood that EMG does not follow a linear relationship to force when contractions are not isometric (fixed muscle length).

2.3.2 Motion Extraction using Electromyography

Musculoskeletal modeling involves creating computational representations of the human musculoskeletal system to simulate movement and estimate muscle forces. These models integrate anatomical data, such as muscle attachment points and joint geometries, with physiological data, including muscle activation patterns derived from EMG. By simulating the interactions between muscles, bones, and joints, musculoskeletal models can estimate the contribution of individual muscles to joint kinematics during various movements [17].

By using parameterized musculoskeletal models, elements of human motion such as range-of-motion, gait positioning, or postural stabilization, can be extracted from EMG data, particularly for standardized movements such as rehabilitation exercises. As shown in Figure 4, shoulder flexion can be represented as a pendulum model with a spring and damper, modeling muscles in the form of a spring who's spring constant is a function of the muscle activation level [18]. Estimates of changes in joint angle also must consider the role of gravity.

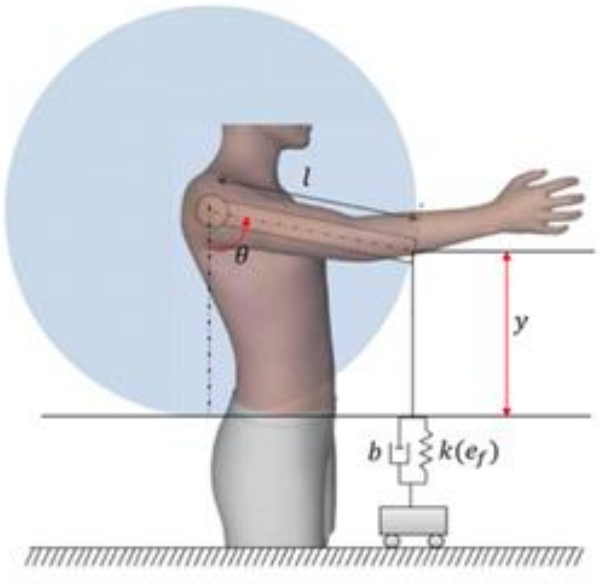

Figure 2-4. Simple pendulum model with spring and damper [18].

By incorporating real-time data from other wearable sensors, such as accelerometers and gyroscopes, researchers can refine and validate their models to better represent dynamic movements and functional tasks [16].

2.3.3 Motion Extraction using Inertial Measurement Units

Several methods exist for measuring human kinematics such as visual estimations, optical motion capture, and inertial-based motion capture. Inertial motor units (IMUs) have increasingly become popular due to their ease-of-use, cost, and versatility [19]. IMUs utilize accelerometers, gyroscopes, and sometimes magnetometers to measure the force, angular velocity, and sometimes magnetic orientation of the device. These measurements can be combined with a kinematic model to accurately estimate elements of human motion. However, there are limitations to IMUs caused by several error sources: inertial sensor drift, linear acceleration interference (e.g. gravitational acceleration), and other electronic interference [19]. Kinematic estimations require integration of measurements, meaning even small errors will accumulate over time, requiring complex algorithms to counteract.

Preprocessing of IMU data typically will include data normalization against a reference frame (such as gravity), noise removal using traditional (e.g. Butterworth) low pass filters, removal of other unwanted artifacts, and data manipulation such as resampling. Then sensor fusion, or the combination of the two or three data types, can be performed with a filter or algorithm. Popular choices for filters for IMU sensor fusion are complementary filters, Kalman filters, and Madgwick filters [20, 21].

Further processing is required to estimate motion elements, such as angular kinematics or spatiotemporal detection, as well as further derivatives such as range of motion. This processing typically requires additional calibration, windowing, and anatomical information such as distance to the joint to perform [20].

3 Time-Domain Sinusoid Fitting for ENOB Estimation

3.1 Introduction

As established earlier, effective number of bits (ENOB) is an intuitive measure of the dynamic performance of an ADC or other data transfer or conversion system. There are two approaches to calculate ENOB, one a frequency domain analysis and the other a time domain analysis.

The frequency domain analysis requires using a discrete Fourier transform (DFT) to convert the time domain signal into the frequency domain. A summation of the power of the input signal frequencies is then compared to the summation of the noise to calculate SINAD. One issue with this approach is that DFTs perform the frequency domain transform over a finite number of frequencies called bins and using a finite signal duration. These limitations cause the spectral estimates to be smeared over multiple bins in the frequency domain, the extent being dependent on bin resolution and the widowing function used by the DFT. This smearing means the a narrow-band sinusoidal input signal is now spread over multiple bins, meaning improper bin selection will affect ENOB calculation by either including noise or excluding signal [22].

The time domain analysis requires fitting a digitally generated sinusoid to the ADC data. The generated waveform is then subtracted from the ADC data, leaving only the noise and distortion, which can be used to calculate SINAD and ENOB [23]. IEEE Standards 1057 [24] and 1241 [25] present methods for sinusoid fitting using three or four-parameter least-squares regression, where the sinusoid is defined as:

$$f(t) = A_0 \cos(\omega_0 t) + B_0 \sin(\omega_0 t) + C_0$$

where ω_0 is the frequency of the input signal and can be plotted in the form:

$$f(t) = A \cos(\omega_0 t + \theta) + C$$

where:

$$A = \sqrt{A_0^2 + B_0^2}$$

$$\theta = \tan^{-1}\left[-\frac{B_0}{A_0}\right], \qquad if A_0 \geq 0$$

$$\theta = \tan^{-1}\left[-\frac{B_0}{A_0}\right] + \pi, \qquad if A_0 < 0$$

The algorithm then finds the values of the parameters which minimize the residual of the following sum of squared differences of a sample with M samples:

$$r_n = \sum_{n=1}^{M} [y_n - A_0 \cos(\omega_0 t_n) + B_0 \sin(\omega_0 t_n) + C_0]^2$$

where r_n is the residuals of the fit, and y_n is the recorded ADC data.

A three-parameter fit (A_0, B_0, C_0) requires a known frequency and is a closed-form solution that will always produce an answer. A four-parameter fit (A_0, B_0, C_0, ω_0) provides a better fit than the three-parameter regression if the actual frequency of the input does not match the known frequency. The four-parameter fit employs an iterative solution that may diverge if a bad initial estimate is provided. To solve the four-parameter regression a non-linear system of equations must be solved in an iterative fashion.

3.2 Methods

The goal was to evaluate the dynamic performance of the ADC and BLE transfer of the nRF58240 by measuring ENOB through time-domain analysis. A four-parameter fitting algorithm was implemented in MATLAB based on the methods presented in [24] and [25]. A copy of the code used can be found in Appendix A.

The initial parameters of the model were set using simple calculations from the noisy signal. The first parameter A_0 was set to the full amplitude of the signal, estimated using the square root of two, multiplied the rms value of the signal. The second parameter B_0 was set to zero, assuming that there is no phase shift in the signal. The third parameter C_0 represents the DC offset, so was set to the mean value of the signal, being zero if the signal has already undergone mean removal. The fourth parameter, ω_0 was set to the fundamental frequency of the signal in radians ($2\pi f$). The fundamental frequency of the signal can be estimated by finding the maximum of the signal's power spectral density (PSD). This is preferable to setting the parameter to the frequency used in function generation to reduce the effect of frequency irregularities from the physical function generator. MATLAB's implementation of Welch's method was used to estimate the PSD.

To solve the regression model, MATLAB's lsqcurvefit() function, which uses a trust-region reflective algorithm to find the parameters which best match a provided dataset, following a user-defined function, initial parameters, and algorithm tolerances. Algorithm tolerances, such as step tolerance, function tolerance, and maximum number of evaluations, define the stopping criteria of the algorithm and can have a significant impact on the results. Step tolerance defines the lower bound of the size of change in each parameter from the last iteration. Function tolerance defines the lower bound on the change in value of the function output from the last iteration. Figure 3-1 shows the application of these parameters as iterations approach a local minimum. MATLAB's default values for both are 10^{-6}, which was found to be sufficient to locate the local minimum of the regression.

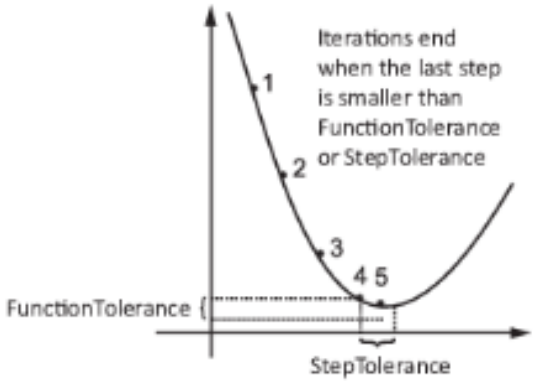

Figure 3-1: Definitions of step tolerance and function tolerance near local minimum [26].

It should be noted that this technique of low-parameter, trust-region reflective modeling is vulnerable to becoming "trapped" in a local minimum that produces a non-ideal fit to the data. However, due to the controlled test conditions this technique is being applied under, with the parameters of the input signal being known to be used for the initial estimate, such a model performed acceptably. Future consideration of alternative models may be appropriate.

3.2.1 Validating Noise Estimation

First, the MATLAB implementation of the time-domain analysis was validated for its accuracy in estimating SINAD. A series of test sine waves with fundamental frequencies of 50 Hz, sampled at 2000 Hz, were generated in MATLAB. White Gaussian noise was then added to the signal using MATLAB's awgn() function to give the signal a specified SNR. Since there is no distortion, SNR can be considered equivalent to SINAD for validation. Tests were run for various added SNR values ranging from 5 to 80 decibels. As shown in Figure 3-2, the MATLAB implementation of the time-domain SINAD estimation can reliably estimate the noise present.

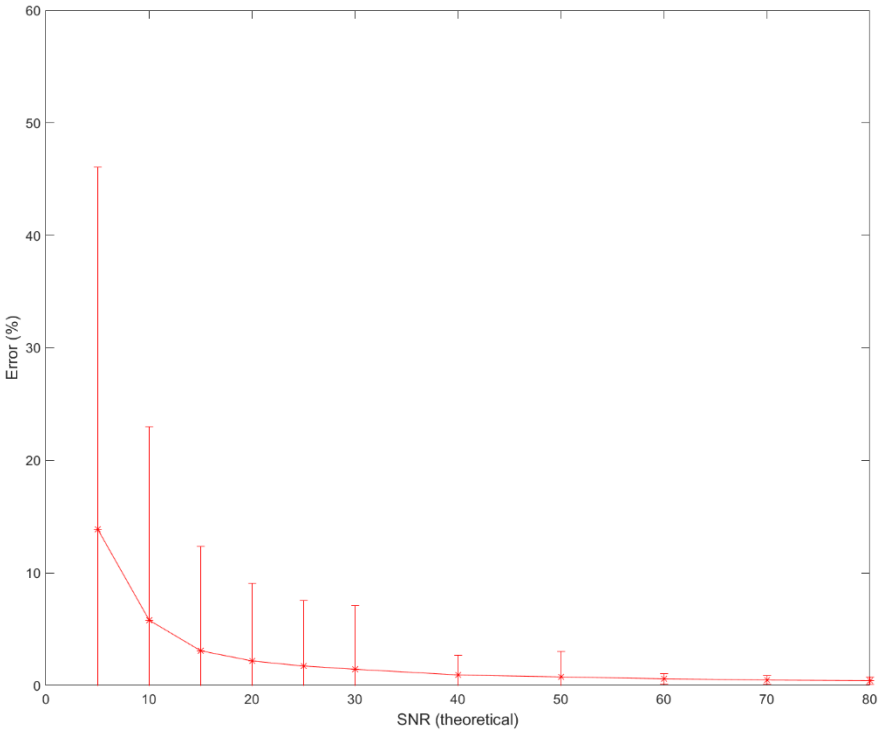

Figure 3-2: Mean percent error ± standard deviation of n = 550 SNR estimations using time-domain analysis.

Statistical testing was conducted to determine whether there were statistical differences over the tested SNR values. To assess the normality of error values, a Kolmogorov-Smirnov (KS) test was used. A KS-test is a non-parametric test used to determine if data follow a given distribution, in this case, a normal distribution. Application of the KS-test, with a 5% significance threshold, rejected the null hypothesis that the data fit a normal distribution for SNR values less than 60dB, so a Friedman test rather than a traditional analysis of variance (ANOVA) test was used to compare the data.

To determine if there were statistical differences in error over the tested SNR values, a Friedman test was applied to the data after being normalized to the theoretical SNR values, with the null hypothesis that the normalized values would be the same. The results of the Friedman test over the entire SNR range rejected the null hypothesis ($p < 2.4954 \times 10^{-69}$). Next, the Friedman test was only applied to the data whose theoretic SNR values were greater than 40 dB, yielding a nonsignificant result ($p = 0.1607$). This shows that while there is a significant difference in the error of noise estimation over the full tested SNR range, the differences are minimal for SNRs above 40 dB.

3.2.2 ENOB Estimation

To evaluate the performance of the system, a peripheral device was set up to read data from a single analog input channel and stream the data over BLE to a central device. The nRF52840-DK (Discovery kit for interfacing with the nRF58240 SOC) was used for both the peripheral and central devices, powered by USB connections to a PC. The ADC input of the peripheral device was configured to have a 12-bit resolution and to sample continuously at 2 kHz with no oversampling. Data was transmitted from the peripheral to the central device every 10 ms, consisting of 4 sync bytes, 8 additional header bytes, and 40 data bytes (2 bytes per sample). From the central node, the data was transmitted set to a PC over a COM port for processing. A custom MATLAB application (see Appendix B) was used to stream data, decode the data packets, and log the data.

Test sine waves with peak-to-peak voltage of 3.44V (selected below maximum input voltage of 3.6V to avoid clipping or attenuation) and frequencies ranging from 50 to 150 Hz were applied to the analog input of the ADC. The test signal was generated by a Hewlett Packard 33120A waveform generator. For each trial, 1 second of data were collected, providing approximately 50 cycles to estimate over, well over the 5 cycle minimum recommended in IEEE standard 1241 [25]. The collected data were then processed using the MATLAB implementation to estimate the ENOB of the system.

Function generators tend to have relatively high levels of noise, data were also collected from the peripheral with the analog input of the ADC shorted to the ground pin of the microprocessor. As a point of comparison, the root mean square (RMS) value of the noise can be taken and used as the NAD value during the calculation of ENOB for the test signals.

3.3 Results

Figure 3-3 shows 5 cycles of data collected from a 50Hz input sine wave, the time-domain fitted wave, and their difference, which is being used as the NAD estimate.

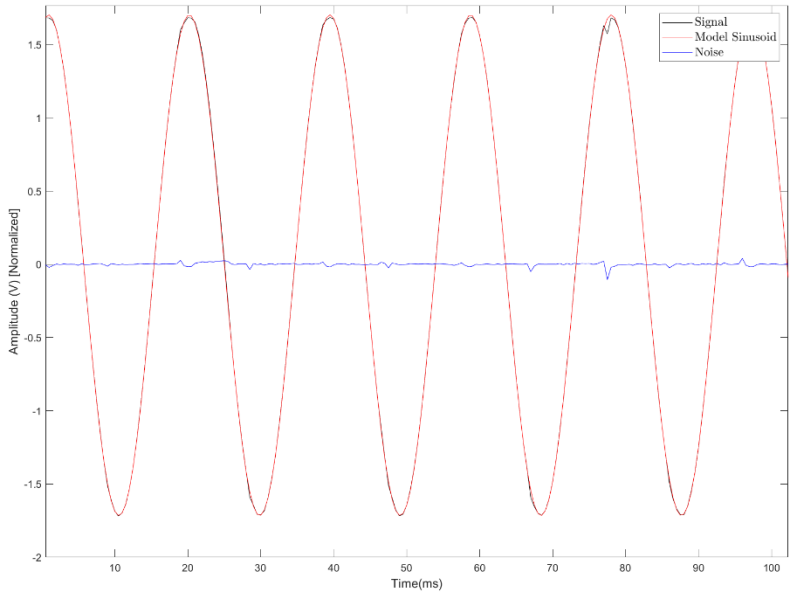

Figure 3-3: 5 cycles of collected 50 Hz sine wave (black, obscured), fitted wave (red), and their difference (blue).

Figure 3-4 shows the power spectral density plots of the time-domain fitted wave alongside the estimated NAD, and noise when the board is shorted. The estimated noise floor is approximately one scale of magnitude higher than the noise when the ADC input is shorted. This suggests that there is a significant amount of noise coming from the function generator that will impact the ENOB estimation. However, there are discernable noise components, such as peaks at harmonics of the signal's fundamental frequency that would be ignored if ENOB is calculated only with the shorted noise. The actual ENOB of the system likely lies somewhere between these two estimates.

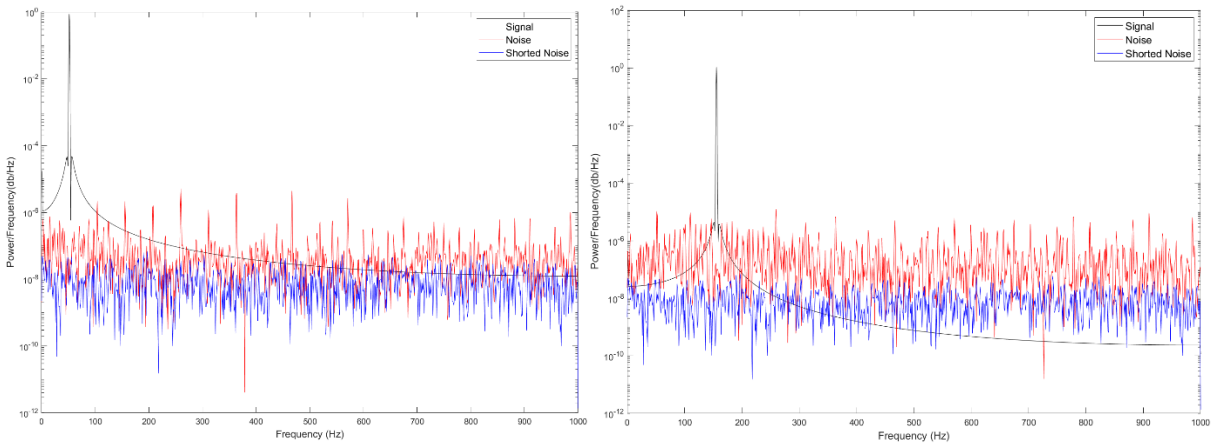

Figure 3-4: Power spectral density of the collected signal and noise at 50Hz (left) and 150Hz (right).

The results (Table 3-1), based on n=5 test sine waves at each of the frequencies: 50, 100, and 150 Hz, showed the effective number of bits of the system is approximately 6.3 bits, decreasing as input frequency increases. A comparison between the PSD plots of 50Hz and 150Hz tests shows increased noise around the fundamental frequency for the 150Hz test. When estimated using the shorted noise the ENOB across all test frequencies can be estimated to be approximately 8.3 bits. Figure 3-5 show the ENOB results plotted against frequency.

Table 3-1: Effective number of bits of the nRF52840 ADC for different test input frequencies. Results based on the time-domain analysis, and the shorted noise floor are shown.

Input Frequency (Hz)	50		100		150	
Trial	ENOB (noise)	ENOB (shorted)	ENOB (noise)	ENOB (shorted)	ENOB (noise)	ENOB (shorted)
1	6.7218	8.3223	6.0578	8.3202	4.5726	8.3319
2	6.677	8.3186	5.3963	8.3241	4.5652	8.3133
3	6.4419	8.3191	5.8241	8.3248	4.3832	8.3228
4	5.8503	8.32	5.3888	8.3178	4.9346	8.3284
5	5.8216	8.3232	6.0378	8.3238	5.6073	8.3269

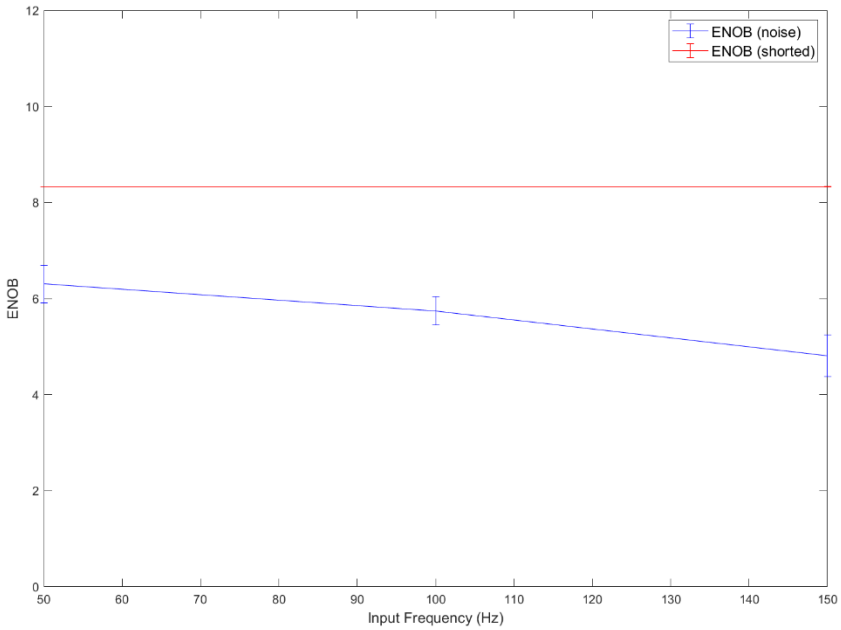

Figure 3-5: Mean percent error ± standard deviation of effective number of bits vs. input frequency.

3.4 Discussion

The evaluation of the dynamic performance of the ADC and BLE transfer of the nRF58240 through ENOB estimation using time-domain analysis provides valuable insights into the system's capabilities and limitations. The implemented four-parameter fitting algorithm in MATLAB based on IEEE standards 1057 and 1241 appears to be capable of estimating SINAD and subsequently ENOB, however, there are limitations of becoming "trapped" in non-ideal local minima. Validation tests using known SNR values affirm the reliability of the time-domain analysis in accurately estimating SINAD across various SNR levels.

Nordic Semiconductor reports an ENOB of 9 bits for the nRF52840 at its maximal sampling rate of 200ksps, 1/1 gain, internal clock, and minimum input resistance. The ENOB of 8.3 bits estimated from the shorted noise floor is a ~7.8% difference from that reference. Further testing could be performed for configurations approaching Nordic's test conditions, and with a non-development kit nRF52840 could provide insights into the specific source of deviation.

The observed decrease in ENOB with increasing input frequency raises questions about the ADC's performance at higher frequencies. However, the consistent results obtained when considering the shorted noise floor suggest that this decrease in performance may be attributed to the noise introduced by the function generator rather than inherent limitations of the ADC itself. The comparison between ENOB estimates using shorted noise and actual collected data underscores the complexity of accurately characterizing system performance in real-world scenarios with multiple noise sources.

In conclusion, while the ENOB estimation process using time-domain analysis offers valuable insights into the ADC and BLE transfer system's capabilities, it is essential to consider potential limitations and external factors such as noise sources. Further refinement of testing methodologies and test conditions could enhance the accuracy and reliability of ENOB estimations, facilitating better informed decisions in system design and optimization.

4 DAQ Precision and Power Consumption

4.1 Introduction

To enhance the effective number of bits (ENOB) in EMG data acquisition, several methods can be employed, including oversampling and increased ADC resolution. Oversampling involves sampling the EMG signal at a rate higher than the Nyquist rate, which can improve resolution and reduce quantization noise. Similarly, increasing the ADC resolution enhances the granularity of the digitized EMG signal, allowing for more precise measurements. The SAADC on the nRF52840 provides options up to 256 times oversampling and resolution options of 8, 10, and 12 bits with a 14-bit format to accommodate additional precision from oversampling.

It is important to note that these enhancements come at the cost in terms of power consumption. ADCs consume power during both the acquisition and conversion phases, only dropping to a low-power state between samples. Oversampling requires increasing the sampling rate, thus reducing the time spent in a low-power state. Similarly, increasing resolution lengthens the conversion phase and increases power consumption.

When using continuous mode on the nRF52840, sampling events are triggered at a given sampling rate. There are two methods in which timing can be implemented, either attaching the sampling event to one of the general-purpose timers, or by utilizing the SAADC's internal timer. Example projects provided by Nordic Semiconductor suggest that the internal timer implementation should only be used for sampling rates above 8kHz. Oversampling can be implemented on the nRF52840 by utilizing an accumulator on the SAADC to average several samples of the analog input. Alternatively, the sampling rate could be increased by the desired factor to be later averaged and downsampled after digital conversion. It should be noted that on the nRF52840 oversampling, can only be used when a single channel is enabled, as the averaging occurs over all enabled channels.

4.2 Methods

The goal was to measure power consumption and the effective number of bits of the nRF52840 under different resolution and oversampling conditions. A peripheral node was set up to read data from a single analog input channel and stream the data over BLE to a central node. The nRF52840-DK was used for both the peripheral and central devices, with the peripheral powered by an external 5V supply and the central by a USB connection to a PC. The ADC input of the peripheral device was configured to have various resolutions and to sample continuously at 2 kHz with various oversampling. Data was transmitted from the peripheral to the central device every 10 ms, consisting of 4 sync bytes, 8 additional header bytes, and 40 data bytes (2 bytes per sample). From the central node, the data was transmitted to a PC over a COM port for processing. A custom MATLAB application (see Appendix B) was used to stream data, decode the data packets, and log the data.

To program the different configurations, Visual Studio Code (v1.72.2 for Windows 64-bit) installed for programming in C/C++ and with extensions for Zephyr was used. Zephyr is an open-source real-time operating system (RTOS) developed by Nordic Semiconductor based on a small-footprint kernel, designed for use on resource-constrained embedded systems. It is integrated into Nordic's software development kit (SDK) along with essential drivers, libraries, and protocol stacks for programming and interfacing with Nordic devices [27]. For this work, nRF Connect SDK v2.1.1 provided through the nRF Connect for Desktop (v4.3.0) was used.

4.2.1 Setting ADC Sampling Rate

Sampling rate can be set by adjusting the timing interval parameters for the timer linked to the ADC sampling event, which also serves as the primary timer for the entire program. The initialization and start command are defined in the main body of the program as shown in Figure 4-1. Lines 783-4 shows the definition of a work queue as a kernel object to process the ADC sampling sequence (adc_sample). The size of the memory block and priority of the work queue can be configured here. Lines 787-8 shows the definition of a timer object which calls the adc_sampling_event, which itself calls the above work queue, at a set time, defined in microseconds (TIMER_INTERVAL_USEC). In this configuration, the timer here defines the sampling rate of the ADC.

```
782
783     k_work_queue_start(&adc_work_q, adc_stack_area, K_THREAD_STACK_SIZEOF(adc_stack_area), ADC_PRIORITY, NULL);
784     k_work_init(&adc_work_info.work, adc_sample);
785
786     // timer
787     k_timer_init(&my_timer, adc_sample_event, NULL);
788     k_timer_start(&my_timer, K_USEC(TIMER_INTERVAL_USEC), K_USEC(TIMER_INTERVAL_USEC));
789
```

Figure 4-1: ADC sampling event timer initialization.

Based on Zephyr's documentation for Nordic ADC's, the sampling rate should also be able to be set by adjusting the interval parameter within the code that defines the ADC sampling sequence. However, adjusting this parameter was not shown to be able to define the sampling rate in the current configuration, likely due to issues with blocking during a given sequence.

4.2.1.1 Other Parameters following Adjusting Sampling Rate

Under the original parameters, when the timer interval was lowered below 500 ns, indicating a sampling rate of 2kHz, data packets (containing 20 samples) could sometimes be split and sent to the central board in two incomplete packets. To address this issue, the peripheral preferred maximum and minimum BLE connection interval must be lowered to accommodate the higher delivery rate. This parameter can be found in the prj.conf file for the peripheral device, as shown in Figure 4-2. Line 28 enables the configuration of the following BLE timing preference parameters. Lines 30-31 set the maximum and minimum

preferred connection interval between the peripheral and central nodes. In this configuration, the maximum interval needed to be lowered to accommodate a 10 ms transmission interval.

```
27  # Set BLE Timing Parameters here
28  CONFIG_BT_GAP_PERIPHERAL_PREF_PARAMS=y
29  # Set the connection interval (connection interval = n*1.25 ms)
30  CONFIG_BT_PERIPHERAL_PREF_MAX_INT=8 #Lowered from 32 to 8 for 2kHz
31  CONFIG_BT_PERIPHERAL_PREF_MIN_INT=8
```

Figure 4-2: BLE timing parameters.

It should be noted that these specific timing parameters were only tested for a single peripheral with a 52-byte packet. Other setups may require different parameters. An alternative method found to address this issue would be to increase the number of samples in each packet, thus reducing the likelihood of overlap between connection intervals.

4.2.2 Setting Resolution and Oversampling

The resolution and oversampling factor of the nRF52840 ADC can be configured from within the devicetree data structure which can interface with the Zephyr libraries through a Visual Studio Code extension. Figure 4-3 shows the default configuration of the nRF52840-DK ADC.

```
81  &adc {
82      #address-cells = <1>;
83      #size-cells = <0>;
84
85      channel@0 {
86          reg = <0>;
87          zephyr,gain = "ADC_GAIN_1_6";
88          zephyr,reference = "ADC_REF_INTERNAL";
89          zephyr,acquisition-time = <ADC_ACQ_TIME_DEFAULT>;
90          zephyr,input-positive = <NRF_SAADC_AIN1>; /* P0.03 */
91          zephyr,resolution = <12>;
92          zephyr,oversampling = <0>;
93      };
94
95  };
```

Figure 4-3: Default ADC initialization in the nRF52840-DK devicetree.

Valid values for resolution (Figure 4-3, line 91) and oversampling (Figure 4-3, line 92) are shown in Figure 4-4, where the value is what should be provided in the devicetree. Resolution is applied before the averaging for oversampling, meaning that for high oversampling, the signal quality improvement will be limited by the resolution.

Value ID	Value	Description
		Set the resolution
8bit	0	8 bits
10bit	1	10 bits
12bit	2	12 bits
14bit	3	14 bits

Value ID	Value	Description
		Oversample control
Bypass	0	Bypass oversampling
Over2x	1	Oversample 2x
Over4x	2	Oversample 4x
Over8x	3	Oversample 8x
Over16x	4	Oversample 16x
Over32x	5	Oversample 32x
Over64x	6	Oversample 64x
Over128x	7	Oversample 128x
Over256x	8	Oversample 256x

Figure 4-4: Configuration options for resolution (top) and oversampling (bottom).

4.2.3 Measuring Current Consumption

Current consumption can be measured from the peripheral device using Nordic Semiconductor's Power Profiler Kit II setup in its ammeter configuration. The kit is connected as an interrupt between the VDD pins (P22) provided on the nRF52840 Development Kit (DK). The Power Profiler application (v4.0.0) provided through the nRF Connect for Desktop (v4.3.0) was used to record measured current from the kit during each trial. Average current was measured over the application's default time window of 10 seconds. A diagram of the configuration for measuring current from a nRF52840-DK with the PPK2 can be seen in Figure 4-5.

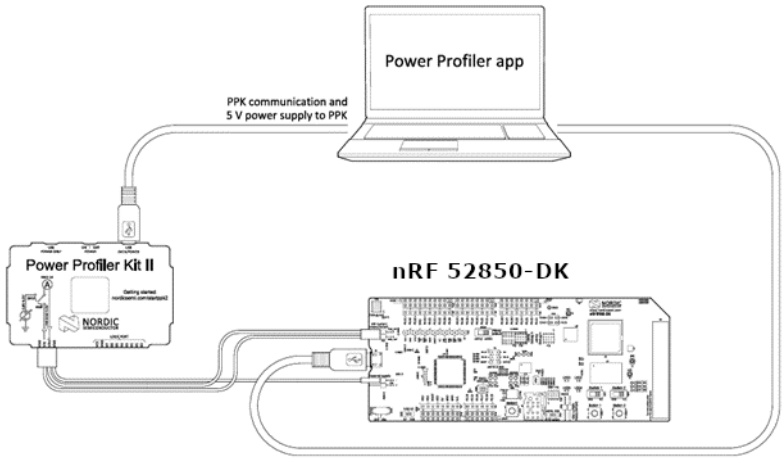

Figure 4-5: PPK2 setup for measuring current on the nRF52840-DK in ammeter mode.

4.2.4 Determining Effective Number of Bits

Simultaneous to measuring current consumption, ENOB was measured using a 50Hz sine wave and the MATLAB implementation presented in chapter 2. Test sine waves with

peak-to-peak voltage of 3.44V (selected below maximum input voltage of 3.6V to avoid clipping or attenuation) were applied to the analog input of the ADC. The test signal was generated by a Hewlett Packard 33120A waveform generator. For each trial 1 second of data was collected, providing approximately 50 cycles for analysis. The collected data was then processed using the MATLAB implementation to estimate the ENOB of the system (see Appendix A).

4.3 Results

4.3.1 Current Consumption

Figure 4-6 shows the results of measuring current consumption from the peripheral device with different resolutions and oversampling factors. Current consumption was shown to decrease by approximately 2%, from 2.16 mA to 2.21 mA, when resolution was changed from 12 to 14-bit configuration. Current consumption was shown to increase by approximately 16.5%, from 2.05 mA to 2.40 mA, when changing from 0 to 16 times oversampling. Current consumption without BLE transfer was approximately 1.8 mA, and with the ADC disabled was 499.5 µA.

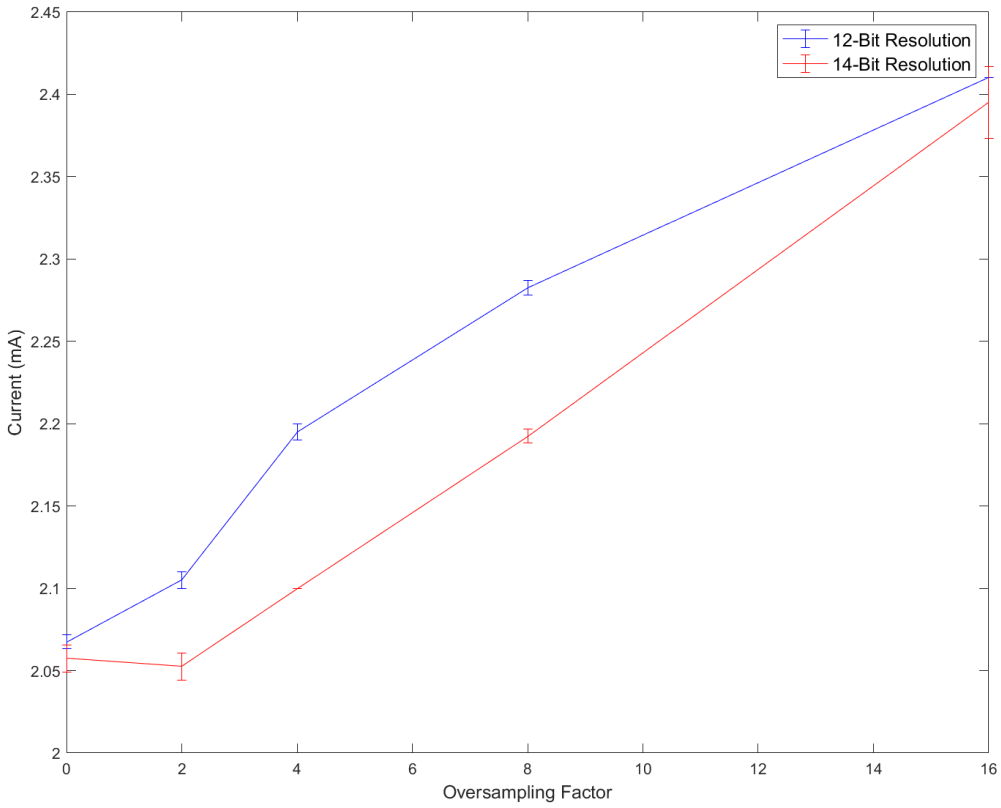

Figure 4-6: Current consumption vs. oversampling factor and resolution.

Statistical testing was conducted to determine whether there were statistical differences in current values over different oversampling and resolution configurations. To assess the normality of current values, a KS-test was used. Application of the KS-test, with a 5% significance threshold, did not reject the null hypothesis that the data fit a normal distribution, so a two-way ANOVA test was used on the dataset.

A two-way ANOVA test was applied on current consumption data grouped by oversampling factor and resolution, with the null hypothesis that there would be no difference in current consumption. An interaction was found ($p = 2.1528 \times 10^{-5}$). Therefore, we initially pursued separate one-way ANOVAs for each resolution. For the 12-bit resolution data, a one-way ANOVA found that there were significant differences in current consumption $[\mathrm{F}(4,95) = 16945, p = 1.3199 \times 10^{-5}]$, and *post hoc* pairwise tests (adjusted for multiple comparisons) found that all oversampling factors differed from each other ($p < 0.0000$). For the 14-bit resolution data, a one-way ANOVA found that there were significant differences in current consumption $[\mathrm{F}(4,95) = 16945, p = 1.3199 \times 10^{-5}]$, and *post hoc* pairwise tests (adjusted for multiple comparisons) found that all oversampling factors differed from each other ($p < 0.0000$) except when comparing oversampling factor 0 to oversampling factor 2 ($p = 0.6598$). Lastly, paired t-tests were conducted within each of the five oversampling factors, with Bonferroni correction (reducing confidence interval to $\alpha = 0.01$), comparing 12-bit to 14-bit resolutions. These tests found that there were significant differences in current consumption $(p < 6.3447 \times 10^{-6})$, except at oversampling factor 16 $(p = 0.0272)$. This shows that both oversampling factor and resolution have a significant impact on current consumption, with evidence that their effects are not independent of each other.

4.3.2 Effective Number of Bits

As shown in Figure 4-7, the ENOB of the ADC increased as the oversampling factor increased, as well as when the resolution configuration was changed from 12 to 14 bits. ENOB was increased by 1.2 bits (~22%) when changing from 0 to 16 times oversampling. ENOB only increased by 0.2 bits (~2.9) over the same range, with significant overlap between the standard deviations in oversampling factors less than or equal to 8.

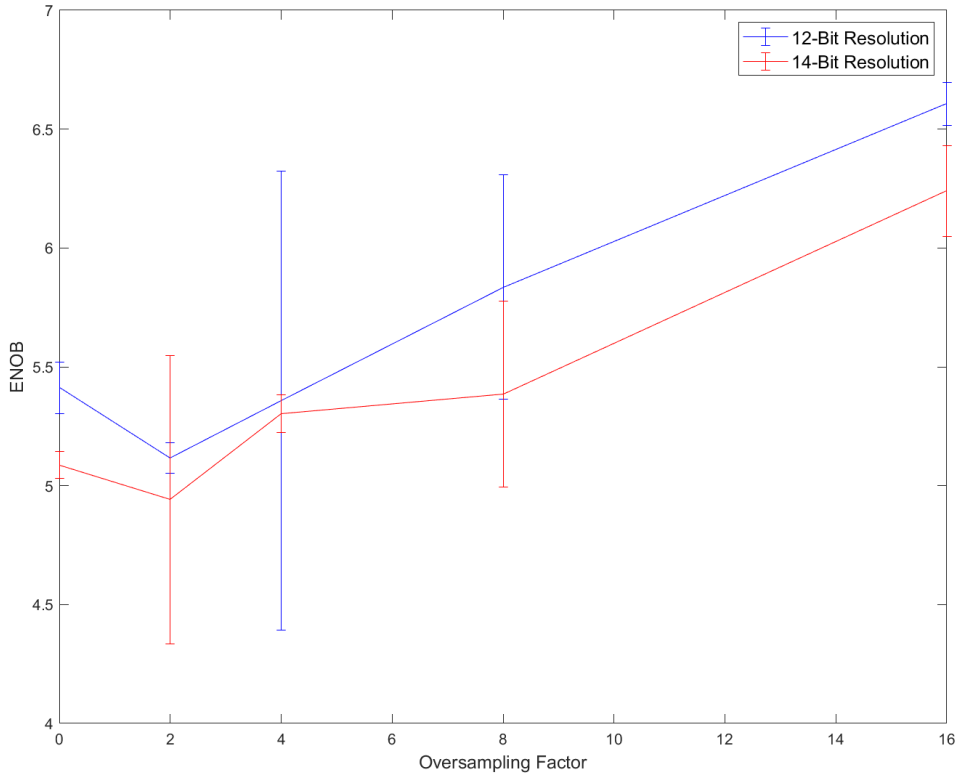

Figure 4-7: ENOB vs oversampling factor and resolution.

Statistical testing was conducted to determine whether there were statistical differences in ENOB values over different oversampling and resolution configurations. To assess the normality of ENOB values, a KS-test was used. Application of the KS-test, with a 5% significance threshold, did not reject the null hypothesis that the data fit a normal distribution, so a two-way ANOVA test was used on the dataset.

A two-way ANOVA test was applied on ENOB data grouped by oversampling factor and resolution, with the null hypothesis that there would be no difference in current consumption, and that there are no interactions between the two factors. The test found no interaction ($p = 0.2514$). The results of the two-way ANOVA test showed that for both oversampling factor $[F(4,190) = 63.82, p = 0.0000]$ and resolution $[F(1,190) = 20.46, p = 0.0000]$, there were significant differences in ENOB. Further post hoc pairwise tests (adjusted for multiple comparisons) found that all oversampling factors differed from each other ($p < 0.0286$), except oversampling factor 0, which did not differ significantly

from oversampling factors 2 and 4 ($p = 0.1442, p = 0.9170$). Post hoc pairwise testing found that 12-bit and 14-bit resolutions significantly differed from each other ($p = 0.0000$). This shows that both oversampling factor and resolution have a significant impact on ENOB, with their impact being independent of each other.

4.4 Discussion

The findings on power consumption demonstrate the trade-offs associated with resolution and oversampling. Unexpectedly, a decrease in current consumption was observed when switching from a 12-bit to a 14-bit configuration. This decrease was relatively marginal (~2%) compared to the substantial increase (~16.5%) observed with higher oversampling factors. The 14-bit resolution does not represent a true change in the ADC's resolution, rather the format in which the data is saved in following oversampling, meaning that the only power consumption difference should be due to memory.

Increasing the oversampling factor led to a notable increase in power consumption. This is consistent with the understanding that oversampling involves acquiring more samples per conversion cycle, thereby increasing the time the ADC spends in active mode, which in turn elevates power consumption. The recorded values provide practical insights for system designers, helping them balance the desired level of oversampling with the available power budget. This is especially important as the ADC appears to make up the majority (~76% at 12-bits, no oversampling) of current consumption during collection.

The ENOB results illustrate the impact of resolution and oversampling on the quality of digitized EMG signals. As expected, increasing the oversampling factor led to a significant, approximately 22%, from 5.2 bits to 6.4 bits, improvement in ENOB. This improvement can be attributed to the reduction in quantization noise and generation noise achieved through the averaging performed by the accumulator. Similarly, increasing the resolution from 12 to 14 bits also contributed to enhancing ENOB, albeit to a lesser extent, approximately 4.8%. This suggests that while higher resolution enables finer granularity in representing signal amplitudes, the benefits in terms of ENOB may not be as pronounced as those achieved through oversampling. This matches expectations of the 14-bit format, where it would not be expected to have a significant impact on signal resolution until precision approaches the limitations of 12 bits.

These findings underscore the importance of careful optimization and parameter selection in EMG data acquisition systems. Designers must strike a balance between achieving sufficient signal quality (as indicated by ENOB) and managing power consumption to ensure the longevity of battery-powered devices. Additionally, the results highlight the need for comprehensive testing and validation to understand the behavior of ADCs under different operating conditions. This knowledge can inform design decisions and enable the development of efficient and robust EMG data acquisition systems tailored to specific application requirements.

5 Python-Based Application for Data Acquisition and Processing

5.1 Introduction

Developments in wireless wearable sensor systems have opened space for potential applications in remote patient monitoring, diagnostics, and real-time feedback for patients. These technologies have the potential to significantly improve patient outcomes by enabling continuous monitoring and personalized interventions, particularly in the realm of rehabilitation and post-operative care [28]. To facilitate the acquisition, processing, and visualization of biometric data an application with a user-friendly graphic user interface (GUI) must be developed.

5.1.1 OrthoKinetic Track Inc

OrthoKinetic Track (OKT) is developing a sensor-based wearable shoulder device for remote patient monitoring of range of motion (ROM) and muscle activity (EMG). The application will utilize biometric data and a predictive machine-learning algorithm to allow real-time guidance to the user for feedback on rehabilitation progress during post-operative recovery.

In this chapter, the development of a python-based application for the acquisition and processing of ROM and EMG data will be discussed. The primary objective of this application is twofold: firstly, to serve as a tool for clinical research during the iterative development of the wearable system, and secondly, to demonstrate the feasibility of creating a user-facing application for future deployment.

5.1.2 Python

Python is a versatile high-level programming language for general purpose coding, commonly used for various applications for its simplicity, readability, and extensive open-source libraries. Several of these open-source libraries were used in the development of the application discussed in this chapter.

Tkinter is a standard Python library for creating GUI applications. It provides a simple and efficient way to develop interactive interfaces, making it ideal for desktop applications. It allows for customizable widgets such as windows, buttons, menus, and other GUI components to enhance user interaction.

Matplotlib is a powerful plotting library that enables the creation of static, animated, and interactive visualizations in Python. It offers a wide range of plotting functions for generating line plots, scatter plots, histograms, bar charts, and more. Matplotlib includes functionality to display a plot within a Tkinter window for real-time and interactive visualization.

Numpy is a fundamental package for numerical computing in Python. It provides support for large, multi-dimensional arrays and matrices, along with a collection of mathematical functions to operate on these arrays efficiently. Numpy's capabilities are crucial for

scientific computations, data manipulation, and algorithm implementations. Scipy is an open-source library that builds upon numpy to provide additional functionality for scientific computing. The scipy.signal module, in particular, offers various signal processing algorithms and digital signal processing (DSP) tools. These tools include filtering, convolution, Fourier analysis, and window functions; facilitating advanced signal analysis and manipulation. Use of this library allows for efficient application of various DSP techniques.

Python's support for object-oriented programming (OOP) enables the creation of complex and reusable data structures. By encapsulating data and methods within objects, developers can organize code more efficiently and facilitate code reuse and maintenance. Commonly used object-oriented data structures in Python include classes, objects, inheritance, polymorphism, and encapsulation. Integrating object-oriented data structures alongside libraries enhances code modularity, scalability, and readability.

5.2 Design Approach

This section will focus on the overall design of the application as shown in Figure 5-1. The main program is responsible for the primary handling of GUI frames, user selection of sub-programs, and handling of data structures that must be passed between sub-programs.

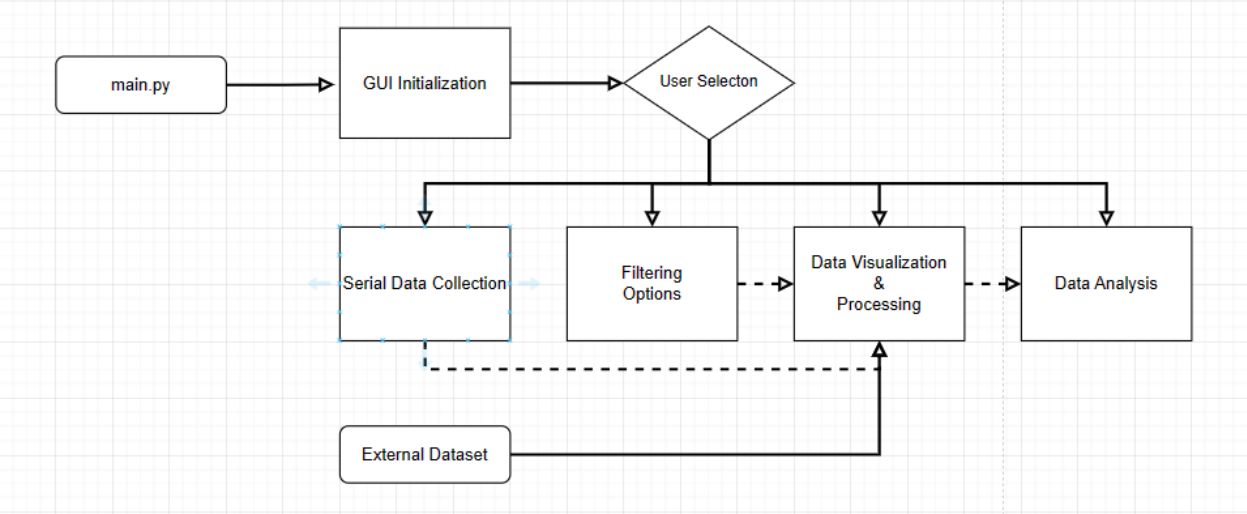

Figure 5-1: Overall system flowchart.

5.2.1 Serial Data Collection

The serial data collection window, shown in Figure 5-2, can be used to read data transmitted to the desktop over its serial communication (COM) port. This allows for the acquisition and real-time visualization of data acquired from systems such as the nRF52840 BLE system explored in chapters 2 and 3.

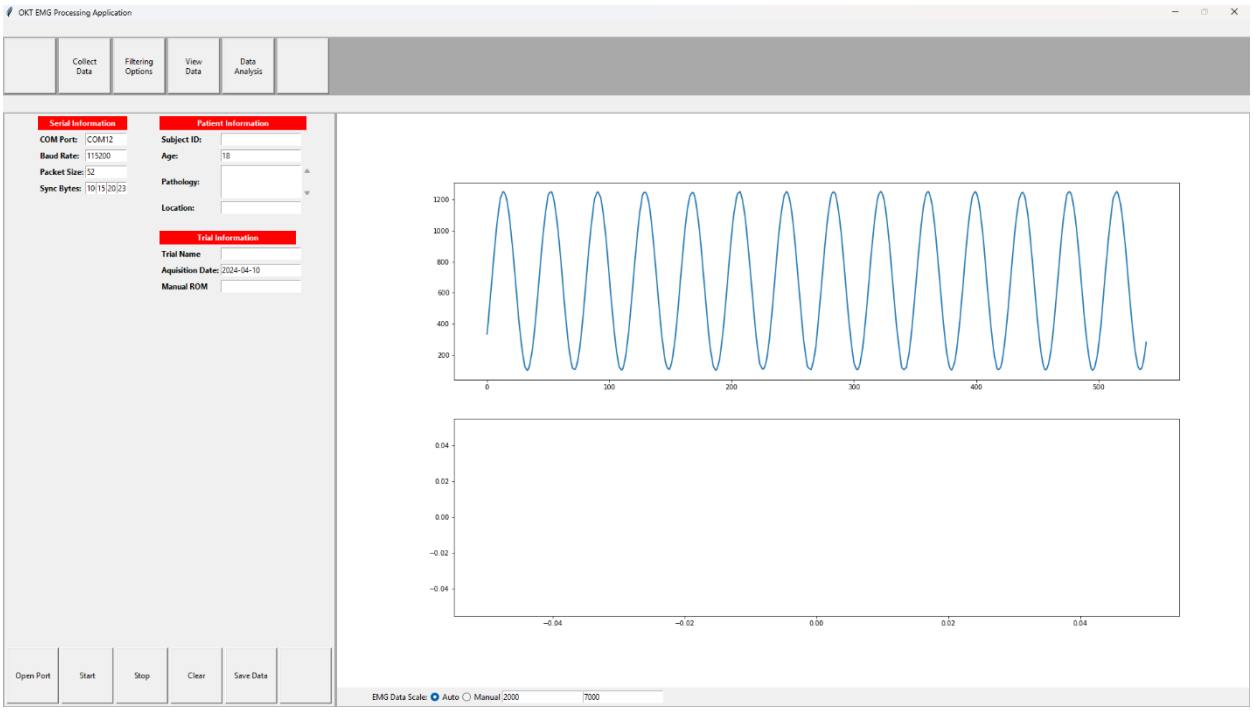

Figure 5-2: Data acquisition window.

The left side of the window (Figure 5-3) includes two user input modules to configure the serial port and define incoming data packets, and to add information about the subject and trial.

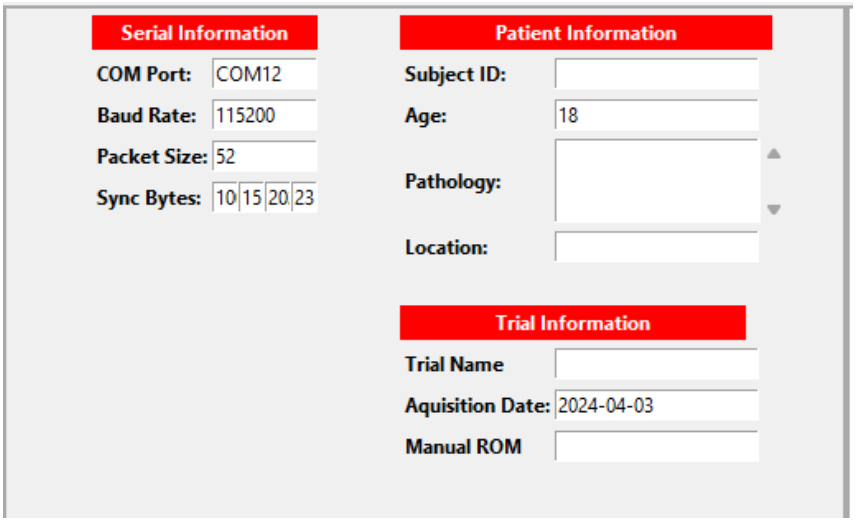

Figure 5-3: Serial, patient, and trial information modules.

To allow for real-time collection and plotting of the data collected over the serial port, the background animation process provided in the matplotlib.animation library was used alongside several interrupts to provide start, stop, and amplitude scaling functionality. The animation function works by repeatedly calling an update module and using the results of it to update a figure. The update function works similarly to the MATLAB script found in

Appendix A, with some adjustments made to adapt it to python and to handle real time acquisition. One notable difference is that rather than acquiring one packet at a time, all bytes available are acquired and placed into a buffer. The buffer is then sorted through to sequentially process full packets within the buffer. When a packet is processed, the data is then added to a frame which the animation function adds to the figure.

Interrupts are handled by the GUI foreground process. The start and stop interrupts change the state of the animation function, and the clear interrupt erases the current figure and stored data. The first time the start button is triggered it initializes the animation before starting. Additionally, the figure's y-axis can be switched between auto-scaling or being set to manual limits. The new selection will only be reflected in the figure upon the next update of the animation.

Data can then be saved to a .csv file using the save data button, which can later be loaded in the data visualization and processing window. The filename is automatically generated based on subject ID and trial information, with the additional information (age, pathology, location, etc.) stored in the header.

5.2.2 EMG Filtering Options

The filtering options window, shown in Figure 5-4, handles the creation of filters for EMG processing in the data visualization window. The options and building of the filters are based on the processing recommendations described in Section 2.1.2 from [4]. The user inputs are then used to construct a new Filter object, which is then added to a Filters object which holds all created filters. A series of different filters can be built in this menu and later individually applied to EMG data channels in the Data Processing and Visualization window.

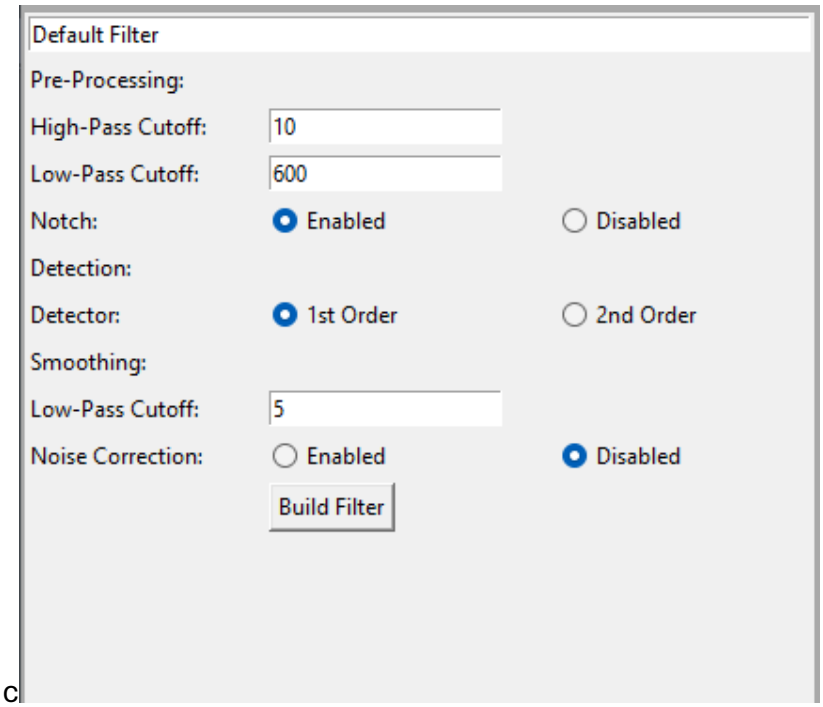

Figure 5-4: Filtering options window.

The Filters class serves as a container for managing multiple filter objects. Upon initialization, it creates an empty list to store Filter objects. The class provides methods to add new filters to the collection and retrieve existing filters based on their names. The add() method allows adding a new filter to the collection. It checks if a filter with the same name already exists; if so, it updates the existing filter with the new parameters; otherwise, it adds the new filter to the collection. The get() method retrieves a filter object from the collection based on its name. The validate() filter can be used to check if the user-inputted parameters fit hard coded requirements. Additionally, the class includes a method named build to call the build() method within the Filter class for all filters contained within its collection.

The Filter class represents a single filter with configurable parameters. Upon instantiation, it initializes with default filter parameters. The define() method allows setting custom parameters for the filter, such as cutoff frequencies and the inclusion of different filtering options. Once the parameters are defined, the build() method constructs the filter using the provided parameters. It utilizes a function called buildFilters() from an external emg processing module. After building the filter, the apply() method can be used to apply the filter to input data. This method employs another function named runFiltering() from the same external module to process the data using the constructed filter. If the filter has not been built before applying, the apply method automatically builds it using a default sampling frequency before applying it to the data.

A flowchart representation of the two classes and their interactions with each other, and external scripts is shown in Figure 5-5.

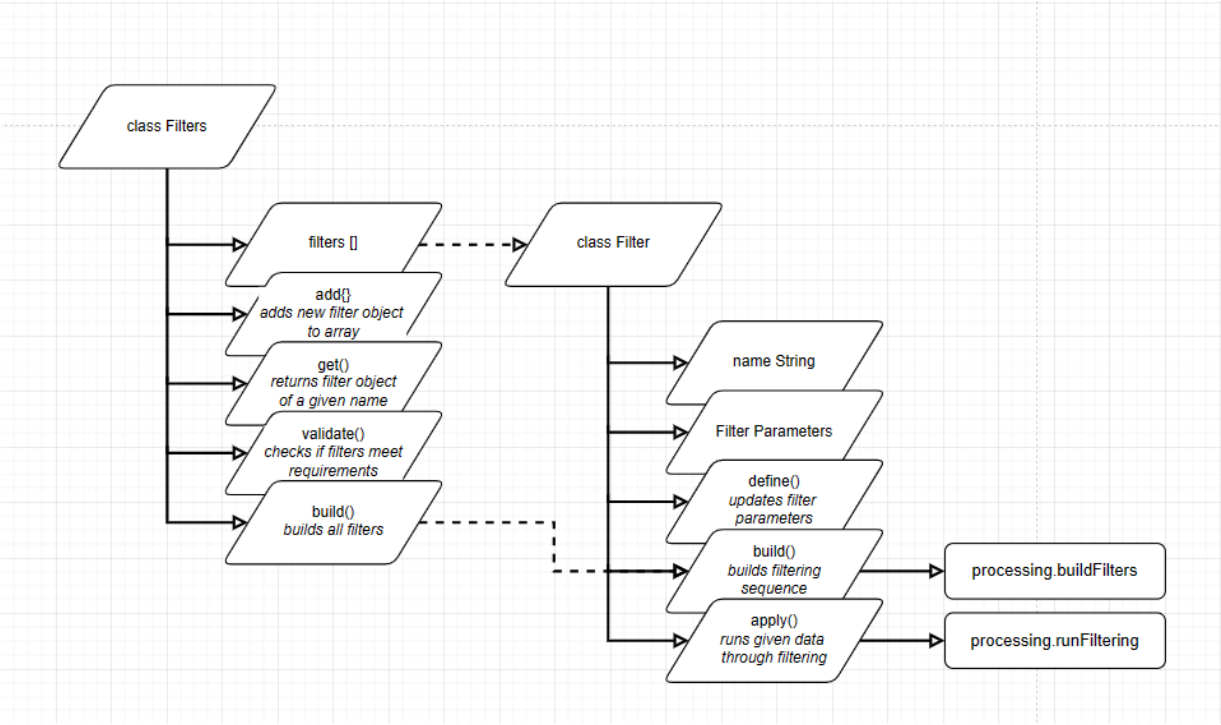

Figure 5-5: Flowchart representation of Filters and Filter class data structures.

5.2.3 Data Processing and Visualization

The data processing and visualization window, shown in Figure 5-7, handles the loading data from a file, storing the data into an easily parsable data structure, the processing of the data with a filter object. The EMG and IMU data shown in Figure 5-7 were collected for assessment of commercially available EMG and IMU sensor hardware, and not for contributions to generalizable knowledge. The data were used herein only to provide display data for the development of the software for the application.

The data structure (Figure 5-6) is organized into a Data class object which serves as a container for a set of Channel objects. The Data class also holds the filename, collection length, and a read() module which when called interprets the file. The Channel class contains the channel name and one Lines object for each datatype: EMG, processed EMG, acceleration, and gyroscope data. The Lines class contains the time series, data, and header(s) as lists, as well as the sampling frequency.

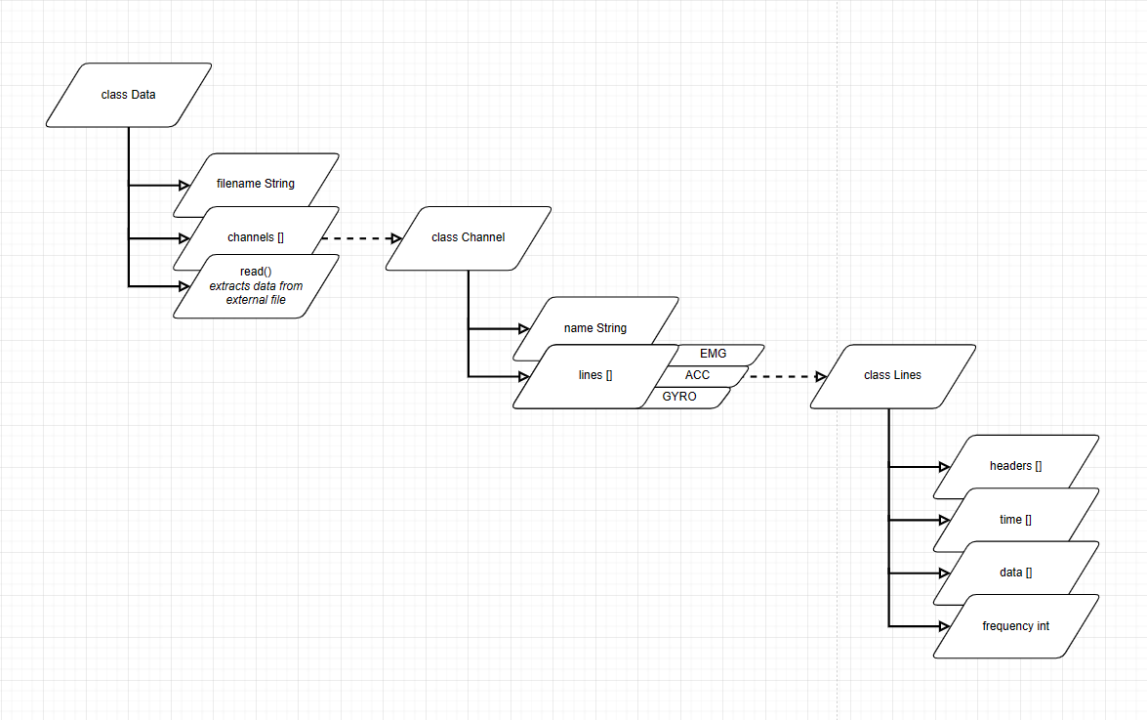

Figure 5-6: Flowchart representation of Data, Channel, and Lines class data structures.

The application currently is able to interpret data files generated by the application's serial data collection, as well as exports from Delsys' EMGworks and TrignoDiscover applications (Delsys, Natick, MA). Since all of these files use the commonly used .csv filename extension, the application currently requires the user to specify the data source. Once the data is loaded, the detected channel names will appear in the channel slots and the EMG and IMU (acceleration and gyroscope) data will be plotted. Filter options can then be selected from the dropdown next to the channel slot. The run processing can be pressed to run through the processing sequence. Currently, this sequence passes a channel of EMG through the Filter.apply() method for the Filter object specified by the dropdown. Processed EMG data is then plotted in the same figure as the raw EMG data and stored within the data structure.

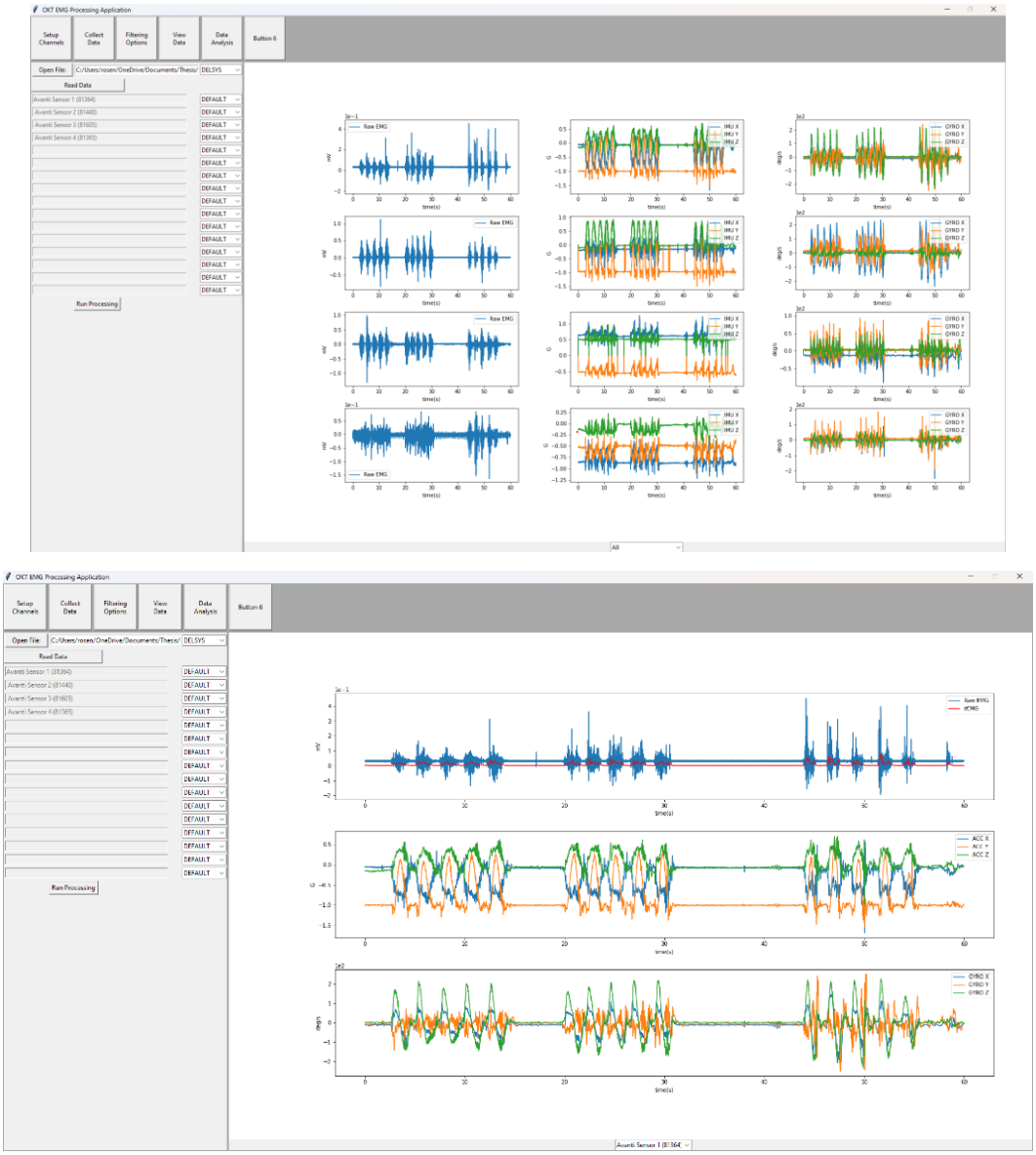

Figure 5-7: Data processing and visualization window displaying the data of all sensors (top) and displaying one sensor with EMG processing (bottom).

5.2.4 Data Analysis

The data analysis window displays options for performing additional data processing, visualization, and analytics beyond filtering. When analyzing range of motion, it is typical to normalize the data to be plotted against the percent of a cycle rather than time. To do this, a detection method must be used to identify individual cycles out of a trial, before normalizing, and averaging over all cycles in the trial (the data shown in Figure 5-7 contain 15 cycles).

To remove high-frequency noise, a 4th order Butterworth filter with a 50 Hz cutoff frequency is applied to the IMU data. The application then uses IMU gyroscope data to estimate the orientation of the IMU in terms of rotational angles (roll, pitch, yaw). Acceleration data is ignored due to complexities to simplify orientation calculations for development. Since there is no standard physiological model being considered at this stage in application development, the linear sum of the rotational angles is considered representative of joint angle during movement. To smooth the orientation estimate, a 4th order Butterworth filter with a 0.5 Hz cutoff frequency is then used.

Since the target application would analyze cyclical "going" and "returning" trajectories, movement cycles are then detected from the orientation estimation signal by identifying peaks using the first derivative of the signal. The detection logic locates two local minimums (negative to positive zero crossings in the first derivative), before checking if a local maximum exists between them which is above a user-defined threshold angle value. Both angle and processed EMG data are segmented at the local minimums as an individual cycle. To average the data over differently sized cycles, the angle and EMG data is resampled to 1000 samples/cycle using a Fourier method implementation, signal.resample() found in the scipy library. To reduce oscillations near the edges of the resampled signal, a windowing function was included in resampling.

The total number of cycles over the trial can then be averaged and displayed in a plot as shown in Figure 5-8. Metrics such as total EMG energy (calculated as the rms of the signal over the cycle) can be reported over the entire cycle or sectioned into going and returning phases of the cycle defined by the maximum angle value over the cycle. The EMG and IMU data shown in Figure 5-8 were collected for assessment of commercially available EMG and IMU sensor hardware, and not for contributions to generalizable knowledge. The data were used herein only to provide display data for the development of the software for the application.

Figure 5-8: Example latissimus dorsi EMG and angle vs. percent cycle during shoulder abduction.

5.3 Discussion and Future Improvements

The described application outlines the development of a python-based application for the acquisition and processing of ROM and EMG data to serve as a tool for clinical research and as a proof of concept for a user-facing application. However, there are several areas that could be further discussed and improved upon for future iterations.

While the application performs real-time data collection, optimizing the efficiency of data handling, memory management and filtering algorithms could enhance performance, particularly for handling large datasets or processing complex filters. The current application was only tested collecting data from one channel for up to three minutes, so further optimization may be needed when using a full array of sensors or prolonged data collection. Additionally, more efficient processing could allow for the implementation of real-time filtering.

Currently, data collected within the application is exported to a generic csv file. In the future data could be exported either in a standard format such as the European data format or with a custom file extension that is easily read by the application. This would additionally allow for the application to easily track previous subjects and trials collected by the application.

Further improvements could also be made to range-of-motion data analysis presented in section 5.2.4. Many filters used for estimating orientation based on IMU data, such as the Madgwick filter, rely upon having a known reference frame to account for bias in the gyroscope and accelerometer measurements. Since measurement device placement varies based on subject and trial, this reference frame becomes hard to account for. More complex algorithms and/or utilizing user-inputted information about the reference frame or physiological model could be used to provide more accurate ROM calculations.

5.4 Conclusion

The development of a Python-based application for the acquisition and processing of range of motion (ROM) and electromyography (EMG) data was outlined in the chapter. This application, developed for OrthoKinetic Track Inc. will be used to assist in the development of a predictive machine-learning algorithm, and wireless wearable sensor system for the shoulder.

The application leverages Python's versatility and the power of various open-source libraries such as Tkinter, Matplotlib, NumPy, and SciPy. These libraries enable efficient data acquisition, processing, visualization, and interaction. Through a series of modular components, the application allows for real-time data collection via serial communication, flexible filtering options for EMG processing, and comprehensive data visualization tools. Furthermore, advanced data analysis techniques, including cycle detection and normalization, contribute to a deeper understanding of patient movement patterns and muscle activity.

While the current version of the application demonstrates promising functionality, there are opportunities for future improvements. Enhancements in efficiency, particularly in handling larger datasets and implementing real-time filtering, could further enhance performance and usability. Additionally, standardizing data export formats and refining algorithms for accurate ROM calculations would increase the application's utility and compatibility with existing systems and protocols.

Overall, the application presented in this chapter will serve as a useful tool for clinical research, development, and as a proof of concept for a future user-facing application for OrthoKinetic Track. The primary objective of this application was to serve as a tool for clinical research during the iterative development of the wearable system and to demonstrate the feasibility of creating a user-facing application.

6 Conclusion

This thesis describes the investigation of the dynamic performance of an analog-to-digital converter (ADC) for wireless wearable systems and the development of a Python-based application for EMG data acquisition and processing.

The first part of the thesis discusses the design of a MATLAB implementation of a time-domain sinusoid fitting technique for effective number of bits estimation. The implemented four-parameter fitting algorithm in MATLAB based on IEEE standards 1057 and 1241 was capable of estimating SINAD and subsequently ENOB, however did have its limitations at higher tolerances, and with non-ideal initial estimations. This technique was applied to the nRF52840 producing a maximal ENOB estimation ~7.8% lower than the specifications documented by Nordic Semiconductor. During testing, noise was measured both from the acquired signal and from the noise floor when the input was shorted to ground. The significant difference between the measured values highlighted the complexity of accurately characterizing system performance in real-world scenarios with multiple noise sources.

The second part of this thesis applies the techniques developed in the first part alongside power measurement of the nRF52840 to explore the impact of different ADC resolution and oversampling configurations. The results illustrated the impact of resolution and oversampling on the quality of digitized EMG signals. As expected, increasing the oversampling factor led to a significant 1.2-bit (~22%) improvement in ENOB. Increasing the resolution configuration from 12 to 14 bits showed lower amounts (~4.8%) of improvement to the ENOB value. Results also showed the increase in oversampling and resolution came at the cost of significant relative power consumption compared to baseline power consumption.

The third part of this thesis discusses the development of a Python-based platform for acquisition and processing of range of motion (ROM) and electromyography (EMG) data. The application was developed with a modular design, utilizing object-oriented coding and open-source libraries, to ease future development.

In totality, this thesis provides insights into the development of wearable wireless sensor systems from analog-to-digital conversion to acquisition, processing, and visualization on a GUI platform.

7 References

[1] R. M. Enoka and J. Duchateau, "Physiology of Muscle Activation and Force Generation," in *Surface Electromyography : Physiology, Engineering, and Applications*, 2016, pp. 1-29.

[2] R. g.-C. o. Ignacio, G.-U. Luis, and M.-T. Armando, "Motor Unit Action Potential Duration: Measurement and Significance," in *Advances in Clinical Neurophysiology*, M. A. Ihsan Ed. Rijeka: IntechOpen, 2012, p. Ch. 7.

[3] D. Farina and A. Holobar, "Characterization of Human Motor Units From Surface EMG Decomposition," *Proceedings of the IEEE,* vol. 104, no. 2, pp. 353-373, 2016, doi: 10.1109/JPROC.2015.2498665.

[4] E. A. Clancy, E. L. Morin, G. Hajian, and R. Merletti, "Tutorial. Surface electromyogram (sEMG) amplitude estimation: Best practices," *Journal of Electromyography and Kinesiology,* vol. 72, pp. 1-18, 2023/10/01 2023, doi: https://doi.org/10.1016/j.jelekin.2023.102807.

[5] C. J. De Luca, "Surface Electromyography: Detection and Recording." [Online]. Available: https://www.delsys.com/downloads/TUTORIAL/semg-detection-and-recording.pdf

[6] E. A. Clancy, E. L. Morin, G. Hajian, and R. Merletti, "Tutorial. Surface electromyogram (sEMG) amplitude estimation: Best practices," *Journal of Electromyography and Kinesiology,* vol. 72, 2023.

[7] K. Rajotte *et al.*, "Power Consumption and Maximum Number of Supported Nodes for BLE Biosensor Applications," in *2023 IEEE 19th International Conference on Body Sensor Networks (BSN)*, 9-11 Oct. 2023 2023, pp. 1-4, doi: 10.1109/BSN58485.2023.10331462.

[8] H. Wang, J. Li, B. E. McDonald, T. R. Farrell, X. Huang, and E. A. Clancy, "Comparison between Two Time Synchronization and Data Alignment Methods for Multi-Channel Wearable Biosensor Systems Using BLE Protocol," *Sensors*, vol. 23, no. 5, doi: 10.3390/s23052465.

[9] S. Bashir, S. Ali, S. Ahmed, and V. Kakkar, "Analog-to-digital converters: A comparative study and performance analysis," in *2016 International Conference on Computing, Communication and Automation (ICCCA)*, 29-30 April 2016 2016, pp. 999-1001, doi: 10.1109/CCAA.2016.7813861.

[10] D. Mercer. "Analog Devices Learning Module 20: Analog to Digital Conversion." Analog Devices. (accessed 2024).

[11] W. Kester. "MT-003 TUTORIAL: Understand SINAD, ENOB, SNR, THD, THD + N, and SFDR so You Don't Get Lost in the Noise Floor." Analog Devices. (accessed 2024).

[12] W. Kester. "MT-001 TUTORIAL: Taking the Mystery out of the Infamous Formula , " SNR = 6 . 02 N + 1 . 76 dB , " and Why You Should Care by." Analog Devices. (accessed 2024).

[13] Nordic Semiconductor, "nRF52840 Objective Product Specification v0.5," Nordic Semiconductor, https://infocenter.nordicsemi.com/pdf/nRF52840_OPS_v0.5.pdf, 2016.

[14] R. De Fazio, V. M. Mastronardi, M. De Vittorio, and P. Visconti, "Wearable Sensors and Smart Devices to Monitor Rehabilitation Parameters and Sports Performance: An Overview," (in eng), *Sensors (Basel),* vol. 23, no. 4, Feb 7 2023, doi: 10.3390/s23041856.

[15] T. J. Roberts and A. M. Gabaldón, "Interpreting muscle function from EMG: lessons learned from direct measurements of muscle force," (in eng), *Integr Comp Biol,* vol. 48, no. 2, pp. 312-20, Aug 2008, doi: 10.1093/icb/icn056.

[16] J. Diong, K. C. Kishimoto, J. E. Butler, and M. E. Héroux, "Muscle electromyographic activity normalized to maximal muscle activity, not to Mmax, better represents voluntary activation," (in eng), *PLoS One,* vol. 17, no. 11, p. e0277947, 2022, doi: 10.1371/journal.pone.0277947.

[17] A. Seth, M. Dong, R. Matias, and S. Delp, "Muscle Contributions to Upper-Extremity Movement and Work From a Musculoskeletal Model of the Human Shoulder," (in eng), *Front Neurorobot,* vol. 13, p. 90, 2019, doi: 10.3389/fnbot.2019.00090.

[18] G. Jang, J.-H. Kim, Y. Choi, and J. Yim, "Human shoulder motion extraction using EMG signals," *International Journal of Precision Engineering and Manufacturing,* vol. 15, pp. 2185-2192, 2014.

[19] M. Rigoni *et al.*, "Assessment of Shoulder Range of Motion Using a Wireless Inertial Motion Capture Device-A Validation Study," (in eng), *Sensors (Basel),* vol. 19, no. 8, Apr 13 2019, doi: 10.3390/s19081781.

[20] B. R. Hindle, J. W. L. Keogh, and A. V. Lorimer, "Inertial-Based Human Motion Capture: A Technical Summary of Current Processing Methodologies for Spatiotemporal and Kinematic Measures," (in eng), *Appl Bionics Biomech,* vol. 2021, p. 6628320, 2021, doi: 10.1155/2021/6628320.

[21] A. Saraf, S. Moon, and A. Madotto, "A Survey of Datasets, Applications, and Models for IMU Sensor Signals," in *2023 IEEE International Conference on Acoustics, Speech, and Signal Processing Workshops (ICASSPW)*, 4-10 June 2023 2023, pp. 1-5, doi: 10.1109/ICASSPW59220.2023.10193365.

[22] T. Girard. Application Note AN95091: Understanding Effective Bits

[23] M. Fonseca da Silva, P. M. Ramos, and A. C. Serra, "A new four parameter sine fitting technique," *Measurement,* vol. 35, no. 2, pp. 131-137, 2004/03/01/ 2004, doi: https://doi.org/10.1016/j.measurement.2003.08.006.

[24] "IEEE Standard for Digitizing Waveform Recorders," *IEEE Std 1057-1994,* p. 0_1, 1994, doi: 10.1109/IEEESTD.1994.122649.

[25] "IEEE Standard for Terminology and Test Methods for Analog-to-Digital Converters," *IEEE Std 1241-2000,* pp. 1-98, 2001, doi: 10.1109/IEEESTD.2001.92771.

[26] *lsqnonlin*. (2022). The MathWorks Inc. [Online]. Available: https://www.mathworks.com/help/optim/ug/lsqnonlin.html

[27] Nordic Semiconductor. "nRF Connect SDK " (accessed 2024).

[28] R. E. Majidi, V., Z. Zhang, E. A. Clancy, and A. Kiapour, "AI-Based Wireless Sensor System for Tracking Muscle Activity and Kinematics of the Shoulder joint," presented at the Orthopaedic Research Society Annual Meeting, 2023, 122.

Appendix A: MATLAB Time-Domain ENOB Estimation

```
clear all; close all; clc;
load('3.31_signal_150_5.mat')
load('3.27_shortednoise.mat')

Y = double(data)/1124;
Y = (Y-mean(Y));
t = 0.0005:0.0005:0.0005*length(Y);
N_short = N_short/1124;

[pxx, f] = pwelch(Y,length(Y),0,length(Y),2000); %Welch's power spectral density estimate
[~, ind] = max(pxx);
fc_hat = f(ind);

%Initial parameter estimates:
A_0 = rms(Y)*sqrt(2);
B_0 =  0;
C_0 = mean(Y);
w_0 = (2*pi*fc_hat);
params_0 = [A_0 B_0 C_0 w_0];

%Function model
fun = @(a, t) a(1).*cos(a(4).*t) + a(2).*sin(a(4).*t) + a(3);
options = optimoptions('lsqcurvefit','FunctionTolerance',1e-20,'StepTolerance',1e-10); lb =
[]; ub = []; %Options for lsqcurvefit
[params_hat, resnorm] = lsqcurvefit(fun, params_0,t,Y,lb,ub,options); %Non-linear data
fitting in least-squares sense

%Generate sinusoid from parameters
A = sqrt(params_hat(1)^2 + params_hat(2)^2);
if A >=0
   Theta = atan(-1*params_hat(2)/params_hat(1));
elseif A < 0
   Theta = atan(-1*params_hat(2)/params_hat(1)) + pi;
end
Y_hat = A*cos(params_hat(4).*t + Theta) + params_hat(3);

NAD_hat = Y - Y_hat;
```

```
figure(1)
t = t*1000;
plot(t,Y,'k',t,Y_hat,'r',t,NAD_hat,'b')
xlabel('Time(ms)')
ylabel('Amplitude (V) [Normalized]')
legend('Signal','Model Sinusoid','Noise','Interpreter','latex')
hold on


SINAD_hat = 20*log10(rms(Y_hat)/rms(NAD_hat));
ENOB_hat = (SINAD_hat - 1.76 + 20*log10(Vio/Vin))/6.02;

SINAD_hat_short = 20*log10(rms(Y_hat)/rms(N_short));
ENOB_hat_short = (SINAD_hat_short - 1.76 + 20*log10(Vio/Vin))/6.02;

figure(3)
[pxx, f] = pwelch(Y_hat,length(Y),0,length(X),2000);
semilogy(f,pxx,'k')
hold on
[pxx, f] = pwelch(NAD_hat,length(Y),0,length(X),2000);
semilogy(f,pxx,'r')
[pxx, f] = pwelch(N_short,length(N_short),0,length(N_short),2000);
semilogy(f,pxx,'b')
xlabel('Frequency (Hz)')
ylabel('Power/Frequency(db/Hz)')
legend('Signal','Noise', 'Shorted Noise','latex')
```

Appendix B: MATLAB Data Collection for Nordic nRF52840

This code was modified based on code developed by Kiriaki Rajotte

```
clear all; clc;

% Read the Data from the Serial Port
num_reads = 5000;
Vio = 3.6; % Maximum ADC Voltage Pk-pk
Vin = 3.44; % Input Voltage Amplitude Pk-pk
%Offset: 0.8V, mean: 1.60V
ADC_noise_rms = 0;

expected_pkt_size = 52;
port = 'COM12';
baud_rate = 115200;


s = serialport(port, baud_rate);
raw_data = [];
pkt_size = [];
for i = 1:num_reads
  cnt = 0;
  while (s.NumBytesAvailable < expected_pkt_size)
    pause(0.0005);
    cnt = cnt + 1;
    %       if cnt > 1000;
    %          break;
    %       end
  end
  raw_data = [raw_data, uint8(read(s, expected_pkt_size, 'uint8'))];

  %raw_data = [raw_data, read(s, expected_pkt_size,'uint8')];

  %data = readline(s);
  i = i + 1
end

array = zeros(num_reads,expected_pkt_size);
for i = 1:num_reads
```

```
  array(i,:) = raw_data(1+(i-1)*expected_pkt_size:expected_pkt_size+(i-
1)*expected_pkt_size);
end

sync_bytes = [100,15,202,235];
data = [];
timestamp = [];
k = 1;

while(k < length(raw_data))
  if raw_data(k:k+3) == sync_bytes(1:4) %checking for sync bytes in raw_data
    disp("Found Sync Byte");
    pkt_size = double(raw_data(k+7)); %the first 8 bytes should contain the pkt_size
    if k+pkt_size-1 > length(raw_data)
      break
    end
    data = [data, typecast(raw_data(k+12:k+pkt_size-1),'int16')]; %concecates the two
bytes into one 16-bit value, adds to data
    timestamp = [timestamp,typecast((raw_data(k+4:k+5)),'uint16')]; %conececates the
two bytes into one 16-bit timestamp
    k = k + pkt_size; %instances a full packet
  else
    k = k + 1;
    if k+3 > length(raw_data) %breaks if there will not be another sync byte
      break
    end
  end
end

X = double(data);
```