Integrated Solutions Towards Wireless Transcutaneous Oxygen Monitor

A Major Qualifying Project (MQP) Report
Submitted to the Faculty of
WORCESTER POLYTECHNIC INSTITUTE
in partial fulfillment of the requirements
for the Degree of Bachelor of Science in

Electrical & Computer Engineering,
Biomedical Engineering

By:

Naisargi Mehta
Olivia Kendzulak
Ryan McSweeney
Ali Attaa

Project Advisor:

Ulkukan Guler

Date: April 2024

Abstract

Respiratory illnesses, such as chronic obstructive pulmonary disease (COPD), lung cancers, and various infections, are leading causes of global mortality [1]. The COVID-19 pandemic has highlighted the critical need for accessible and effective monitoring technologies outside traditional hospital settings [2–4]. In response, our project has worked with a novel noninvasive transcutaneous oxygen monitor that measures the partial pressure of oxygen through the skin $PtcO_2$, which correlates with arterial oxygen levels(PaO_2). This monitor is specifically designed for use in non-clinical environments, enhancing its applicability for everyday monitoring. A significant challenge in such settings is motion artifacts, which can severely affect the accuracy of sensor readings. To address this, our study implemented three testing protocols—Single Motion Test, Dual Motion Test, and Extended Motion Test—to evaluate the sensor's resilience against motion-induced inaccuracies. These tests simulate various real-world scenarios, providing insights into the sensor's performance and identifying necessary corrective measures to maintain accuracy. Additionally, we incorporated Bluetooth Low Energy (BLE) technology and developed a Graphical User Interface (GUI) to improve the monitor's wearability, data transmission efficiency, and user interface. These advancements have not only enhanced the robustness and user-friendliness of our monitoring system but also represent a significant step forward in remote patient monitoring, enabling reliable oxygen level assessments even during physical activities.

Acknowledgements

We would like to acknowledge Parisa, Saadatmand Hashemi and Isil Isiksalan for their generous contributions in guiding and reviewing this work, as well as Vladimir Vakhter, Burak Kahraman, and Tuna Tufan for their valuable support, which greatly contributed to the success of this research.

Contents

List of Tables

List of Figures

Abbrevations

Abbreviation	Full Form
AAE	Average Absolute Error
ADC	Analog to Digital Converter
AFE	Analog Front End
ANC	Active Noise Cancellation
API	Application Programming Interface
BLE	Bluetooth Low Energy
CLI	Command Line Interface
COPD	Chronic Obstructive Pulmonary Disease
FDA	Food and Drug Administration
FFT	Fast Fourier Transform
FUS	Firmware Upgrade Service
GATT	Generic Attribute Profile
GUI	Graphical User Interface
IoT	Internet of Things
LED	Light Emitting Diode
MCU	Microcontroller Unit
MWSCAS	Midwest Symposium on Circuits and Systems
NICU	Neonatal Intensive Care Unit
NLMS	Normalized Least Mean Squares
PaO_2	Partial Pressure of Oxygen in Arteries
PPG	Photoplethysmography
PO_2	Partial Pressure of Oxygen
PSD	Power Sprectral Density
$PtcO_2$	Transcutaneous Partial Pressure of Oxygen
STFT	Short Time Fourier Transform
STM	STMicroelectronics
b SVD	Singular Value Decomposition
SoC	System on Chip
SSA	Singular Spectrum Analysis
TOM	Transcutaneouss Oxygen Monitor

1 Executive Summary

Respiratory illnesses, including chronic obstructive pulmonary disease (COPD), lung cancers, and various infections, are significant contributors to global mortality [1]. The COVID-19 pandemic has underscored the urgent need for accessible monitoring technologies that can operate effectively outside of traditional hospital settings [2–4]. Monitoring the partial pressure of oxygen in arteries (PaO_2) can help with the early detection of pulmonary diseases, such as COPD, COVID-19, and lower respiratory diseases [6]. In response, a luminescence-based sensor [7] has been developed to monitor transcutaneous oxygen ($PtcO_2$), a measure of the oxygen diffusing through the skin, which reflects PaO_2. Our project has contributed in the development of this sensor for transcutaneous oxygen monitoring, improving its ability to function seamlessly in non-clinical environments.

Outside the clinical settings, the reliability and accuracy of sensor readings are crucial. Motion artifacts, caused by any movement during the monitoring process, can greatly degrade the quality of light-based sensors [8–13]. Recognizing the importance of this issue, our project has conducted a thorough assessment of how motion artifacts impact our sensor's performance. To rigorously test and enhance the robustness of our sensor against motion-induced inaccuracies, we have structured our investigation into three distinct tests: single motion testing, dual motion testing, and extended motion testing. Each test is designed to simulate different real-world scenarios where motion could influence the sensor's readings. By analyzing the outcomes of these tests, we aim to understand the conditions under which the sensor maintains its accuracy and to identify scenarios where corrective measures may be necessary. The insights gained from these experiments are key in enhancing our understanding of the sensor's resilience and guide our development of strategies to mitigate the effects of motion artifacts in non-clinical environments.

In addition to our focused testing protocols, we have integrated Bluetooth Low Energy (BLE) technology to enhance the monitor's wearability and data transmission efficiency. Furthermore, to improve accessibility and usability, we have developed a graphical user interface (GUI). This interface facilitates serial data transmission with the Transcutaneous Oxygen Monitor (TOM) board, runs various sensor tests, and allows for the configuration of sensor settings. This GUI enhances the user experience by ensuring that adjustments and monitoring can be conducted with ease and precision. By combining innovative sensor technology with advanced data management and wireless communication capabilities, our system represents a significant advancement in remote patient monitoring. It offers a robust, user-friendly solution tailored to meet the dynamic needs of the users, ensuring precise and reliable oxygen level monitoring, even during physical activity.

2 Introduction

Respiration is a fundamental physiological process vital for cellular function, regulation of blood pH, and maintaining overall homeostasis. According to the World Health Organization, some respiratory illnesses (COPD, lower respiratory infections, trachea, bronchus, lung cancers) are among the top ten diseases that cause death worldwide [1]. The management of these illnesses heavily relies on oxygen monitoring to track their progression. This necessity, highlighted during the COVID-19 pandemic, underscores the need for effective respiratory monitoring solutions easily deployable outside of the clinical setting [2–4]. As a result, remote patient monitoring emerges as a significant concern [14].

Monitoring tools that provide real-time data can significantly enhance the management of respiratory illnesses, offering early detection that is essential for timely and effective treatment. Monitoring PaO_2 can help with the early detection of pulmonary diseases, such as COPD, COVID-19, and lower respiratory diseases [6]. A recent luminescence-based sensor [7] measures oxygen diffusing through the skin, also known as transcutaneous oxygen ($PtcO_2$), which is correlated with PaO_2. This method uses a light emitting diode (LED) to excite a luminescent sensing film, making luminophore molecules transfer from their ground energy state to a higher energy state [15]. As the luminophore molecules revert from the higher energy state, they emit light characterized by intensity and lifetime (τ). Both luminescence intensity and lifetime correlate inversely with the partial pressure of oxygen (PO_2). Compared to intensity-based methods, the lifetime-based approach is more robust against optical path changes (i.e. motion artifacts) and excitation strength [16]. To calculate the lifetime, the light detected by a photodiode is processed through an analog front end (AFE) and converted to digital form with an analog to digital converter (ADC) [17].

In the dynamic and ever-changing day-to-day environment, ensuring the reliability and accuracy of sensor readings is crucial. Motion artifacts, which can arise from any movement during the monitoring process, significantly impact the quality of sensor signals. Consequently, conducting thorough tests to understand the sensor's behavior under motion conditions is invaluable. This testing not only reveals potential weaknesses but also guides us in implementing necessary corrections to enhance sensor performance. Recognizing the significance of our findings, we have submitted a detailed paper on our motion artifact testing methodologies and results to the Midwest Symposium on Circuits and Systems (MWSCAS). Moreover, we support continuous wireless operation of the sensor with the integration of BLE technology for wireless data transmission [17]. Furthermore, to effectively utilize this prototype, a reliable GUI is developed for automating data collection and storing data in the cloud via Google's application programming interfact (API), enhancing the overall user experience [14]. The GUI facilitates adjustments to sensor settings, such as those

for the temperature sensor, accelerometer, and AFE, enabling straightforward modification of system blocks.

3 Project Overview

The endeavor of developing an integrated solution for a wireless transcutaneous oxygen monitor is a multidimensional project aimed at addressing the critical need for continuous, accurate monitoring of oxygen levels on the skin surface. This Major Qualifying Project (MQP) is structured around several core components, each designed to overcome specific challenges in the realm of wearable health monitoring technologies. These components include the development of a correction algorithm, the implementation of motion artifact testing, the incorporation of BLE for wireless data transmission, and the development of a user-friendly GUI.

3.1 Quantifying the Effects of Motion Artifacts

A comprehensive testing framework was established to quantitatively assess the impact of motion artifacts on sensor readings. This involved creating protocols for single, dual, and extended motion tests under various oxygen pressure levels. The objective was to simulate real-world conditions under which the sensor's resilience to motion artifacts could be thoroughly evaluated. These tests were significant in validating the need for a correction algorithm.

3.2 Correction Algorithm

The key to improving accuracy and reliability in the oxygen monitoring device is the utilization of a robust correction algorithm. This algorithm is designed to counteract the effects of motion artifacts, which are prevalent issues in wearable devices that can greatly affect the precision of measurements. Our research advocates for the adoption of the Wiener Filter technique to preprocess the sensor data, effectively enhancing signal quality by reducing the noise generated by motion artifacts. Additionally, we recommend employing a hybrid approach by applying another filtering technique to further refine data accuracy and reliability.

3.3 BLE Development

The introduction of BLE technology marks a significant advancement in the project's goal of achieving wireless connectivity. This development facilitated the transmission of data to external devices, thereby enabling real-time monitoring without the constraints of wired connections. The BLE framework was care-

fully designed for the oxygen sensing application, achieving organized data transfer and remote sensor configuration.

3.4 Graphical User Interface Implementation

To enhance user interaction with the transcutaneous oxygen monitor, a graphical user interface was developed. This interface provides a visual representation of the sensor data, offering an intuitive platform for users to engage with the device. The GUI development focused on usability and accessibility, aiming to make the monitoring process as straightforward as possible for a diverse user base.

4 Background

4.1 Transcutaneous Oxygen Sensing

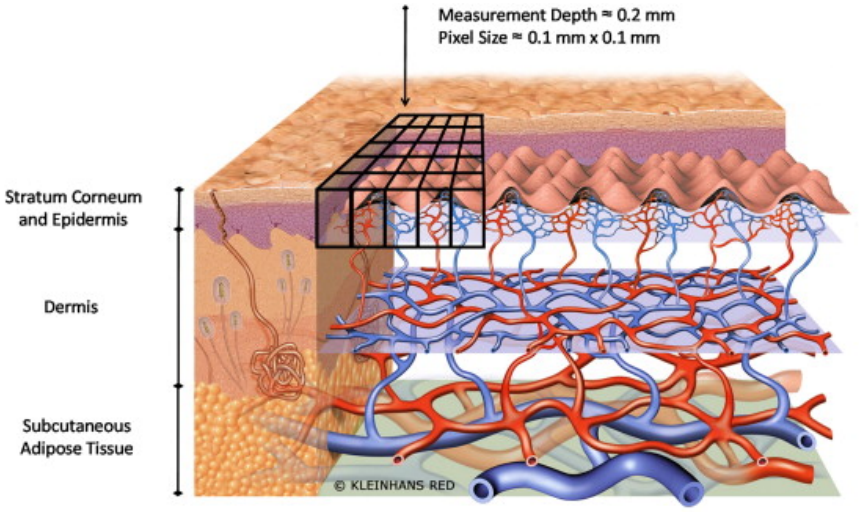


Figure 1: The Vascular Anatomy of the skin, highlighting the different layers, and densities of veins and arteries across them [5].

The circulatory system serves as a crucial mechanism for delivering oxygen to tissues throughout the body, ensuring their proper functioning [18]. The process begins with oxygenation in the lungs, where blood undergoes a transformation, becoming oxygen-rich. This oxygenated blood then embarks on its journey through the intricate network of blood vessels, as visualized in Fig. 1 [5]. Micro-circulation, which encompasses the smallest blood vessels such as arterioles, capillaries, and venules, plays a pivotal role in this journey. Capillaries, stemming from arteries, serve as the primary conduits for oxygen delivery to tissues. As blood courses through these capillaries, oxygen molecules diffuse from the blood into the surrounding tissues, nourishing them and facilitating various metabolic processes [19]. Notably, the arteriole end of the capillary exhibits the highest concentration of oxygen within the bloodstream, ensuring a robust supply to the adjacent tissues. This process underscores the intricate dance between the circulatory system and tissue oxygenation, highlighting the importance of efficient oxygen delivery for the body's vitality and health.

4.1.1 Factors Affecting Transcutaneous Oxygen

Transcutaneous oxygen is a measure of the oxygen being supplied to the skin from the surrounding and underlying tissue [20]. Therefore, it is impacted by multiple different factors that change the rate of oxygen diffusion:

- **Dermal Perfusion**: The blood flow to the dermis. The oxygen pressure in the dermis correlates closely with the levels of oxygen in the arteries. Adequate blood perfusion and oxygen diffusion from the blood are necessary for transcutaneous oxygenation. [21–23].

- **Epidermal and Subcutaneous Fat**: The thickness of the fat layers on the epidermis and subcutaneous level hinder the diffusion of oxygen as it increases the diffusion distance [24].

- **Capillary Density**: The number of capillaries can affect the $PtcO_2$ levels as the level of perfusion across capillaries directly modifies diffusion distance for oxygen [25, 26].

Recognizing the significance of these factors is crucial in determining the optimal placement of the transcutaneous oxygen monitor and the most appropriate timing for its use. Factors such as dermal perfusion, epidermal and subcutaneous fat thickness, and capillary density directly influence the diffusion of oxygen and, consequently, the accuracy of transcutaneous oxygen measurements. By considering these variables, we can understand the relevant factors that affect the measurement of $PtcO_2$ as well as assess the best places to take such measurements.

4.1.2 Transcutaneous Oxygen Monitor Techniques

To ensure that the measurements acquired are reliable measures of the oxygen pressure, the TOM sensor must be placed at a reliable location. Preferred sites for transcutaneous oxygen measurements are typically those with thin skin and a rich blood supply [27]. A study indicated that the palm of the hand yields the most consistent transcutaneous readings, likely due to its high vascularization [28]. Other viable locations, such as the forearm, chest, and abdomen, have been identified for testing; however, these areas were assessed under conditions of applied heat to enhance blood flow and diffusion. Conversely, research using a transcutaneous monitor revealed that the anterior chest may not be optimal, exhibiting lower readings compared to the hand and arm [29]. Infants, including those in neonatal intensive care units (NICUs), often require different measurement sites. For NICU patients and infants, the upper chest is commonly utilized. Additionally, locations like the ear have been explored due to their lower risk of detachment during movement [27]. For our research, testing was done on the forearm as well as a method that utilized a robot to mimic erratic movements, as demonstrated in Section 6.

4.2 Wiener filter

The Wiener filter is a classic algorithm that has various usages and implementation varieties, especially in the realm of denoising data. It performs the minimum mean square error estimation of the desired signal given another related process [9]. In this implementation, a windowed wiener filter is utilized, and it operates within a moving window of fixed size on the signal. At each step, it recalculates the filter based on the latest measurements within the current window. This adaptability allows it to efficiently handle non-stationary noises and interference. As the window slides along the signal, the filter adjusts dynamically. The continuous updating ensures that the filter remains relevant even as the signal evolves. Unlike the classic Wiener filter, which may require an infinite amount of past and future data, the windowed version strikes a balance. It uses only input data within the window, making it more suitable for real-time applications [30].

Motion artifacts often introduce noise at specific frequency components, which can be distinct from the frequency content of the photoplethysmography (PPG) signal itself. The Wiener filter operates in the frequency domain, allowing it to selectively attenuate noise components in the spectrum. This means that frequencies that are most affected by the noisy signal are given less importance.

The Wiener filter takes the two inputs to its signal the sensor data and and the motion data incoming from the accelerometer:

$$X(f) = S(f) + N(f) \tag{1}$$

Where X(f) represents the total incoming signal, S(f) and N(f) represent the sensor and motion data, respectively. Then, the power spectral density (PSD) of each of the signals is calculated by first taking an STFT. Unlike a normal fast Fourier transform (FFT), the short-time Fourier transform (STFT) offers frequency information localized in time, suitable for scenarios where signal frequency components change over time. In contrast, the conventional Fourier transform yields frequency details averaged across the entirety of the signal's period [30].

$$P_{xx} = |X(f)|^2 \tag{2}$$

$$P_{nn} = mean(|N(f)|)^2 \tag{3}$$

The PSD is part of the filter that gives it adaptability as it shows the most affected frequencies and uses those to construct the filter coefficients using an adaptive transfer function H_{min} and scale factor K_{OVER}. These coefficients are then applied to the signal and translated back into the time domain via an Inverse

STFT. This signal is then used as the estimated clean signal and filters out the values that are out of the set in this specific window.

4.3 Bluetooth Low Energy Techniques

In the evolving landscape of wireless technology, Bluetooth stands out as a pivotal innovation, reshaping how devices communicate over short distances. However, despite its widespread use and benefits, there are inherent limitations and challenges that affect its application, particularly in security and power consumption.

Bluetooth technology, especially Bluetooth Low Energy (BLE), is central to the operation of many personal devices today [31]. While BLE is designed to reduce power consumption, making it ideal for wearable and sensor-based applications, it is not without its issues. For instance, BLE's power efficiency, though superior to classic Bluetooth, still poses challenges in environments requiring constant data transmission over extended periods. Additionally, its security protocols, while robust, are still vulnerable to attacks that could compromise data integrity and privacy.

The architecture of BLE involves several key components: Generic Attribute Profile (GATT), Generic Access Profile (GAP), services, and characteristics. GAP and GATT play fundamental roles in BLE communication, where GAP controls device connections and advertising, and GATT acts as the backbone through which data is structured, stored, and exchanged between BLE devices. GATT operates on the principle of a server-client model, where the GATT server stores data and makes it accessible to the client upon request using a BLE protocol stack. The GATT protocol dictates how data is formatted and transferred over BLE's low-energy protocol through the use of services and characteristics, which are similar to objects and their methods in an object-oriented programming structure [32–34].

BLE extends the utility of Bluetooth by enabling wearable devices to remain operational on a single battery charge for long periods. This is achieved through its design to facilitate small bursts of data transmission, reducing power usage without sacrificing the communication range. The implications of such a feature are profound, especially in the domain of sensor development, where continuous data collection and transmission are essential, yet power resources are limited [35].

4.4 GUI Techniques

Graphical User Interfaces GUIs have transformed how humans interact with computers, making the computing experience intuitive and user-friendly. Originating in the 1970s, GUIs marked a pivotal shift from the command-line interfaces prevalent at the time. They present digital information through visual elements like windows, icons, menus, and buttons, which simplify complex tasks, streamline workflows, and enhance productivity.

This visual format allows users of varying technical proficiency to navigate software applications easily, without needing extensive knowledge of programming languages or system operations. The impact of GUIs spans multiple sectors, from personal computing where they are essential for tasks like web browsing and multimedia consumption, to industrial settings where they help manage and control complex machinery. In every context, GUIs enhance the user experience by providing a cohesive and efficient interface for interacting with digital systems.

4.4.1 Python GUIs and Tkinter

The Python development environment was utilized for the GUI. Python is generally regarded as a high-level scripting language. Python is a large language and due to it being general-purpose and cross-platform, it is very accessible.

Multiple libraries within Python allow for constructing GUIs. Notably, TKinter, PySide, and PyQT. Each of these libraries offers unique features and styles that are advantageous in their own right. While PyQT offers a more comprehensive way of modifying and creating GUIs and applications in Python, it's downfall comes from the fact that it is not a free-to-use environment and requires obtaining external licensing [PyQt requires paying USD 550 per developer]. However, PyQT improves upon TKinter in its ability to build projects that are also compatible with mobile users making it useful for future considerations. Alternatively, PySide is a newer library that is free since it is licensed under the GNU Lesser General Public License states it is free to use, unlike PyQt which is in the GNU General Public License that requires paying for commercial or personal use. Both these libraries require utilizing the Qt framework and environment which are separate from the main coding environment used in VSCode. One of the other arguments for using a different environment over TKinter is for its retro look, as it looks outdated. However, to combat this, TKinter.ttk was introduced, which modernized the interface at the expense of configuration capabilities (Figure 3).

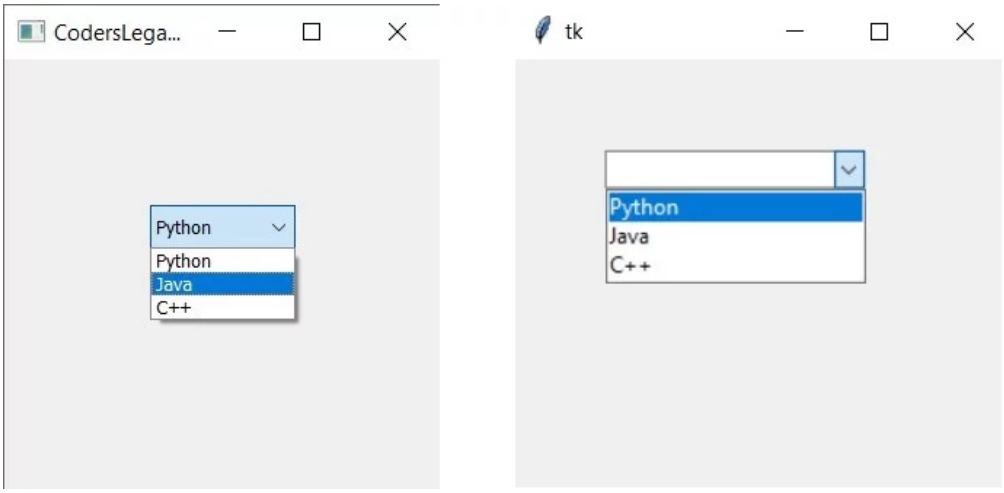


Figure 2: Comparison between a PyQT widget (Left) and TKinter widget (Right

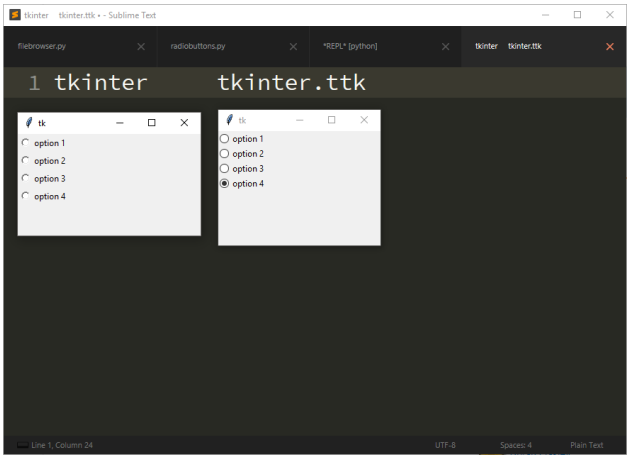


Figure 3: Comparison of the GUI elements by using the stock TKinter (left), and TKinter.ttk (right

Python's simplicity and versatility make it an ideal choice for developing GUIs, especially using the Tkinter library. As a standard Python library, Tkinter facilitates the creation of desktop applications through an intuitive widget hierarchy. Developers can easily place and customize widgets such as buttons and entry fields within windows, tailoring their appearance and behavior to suit specific needs.

Tkinter integrates seamlessly with Python's event-driven model, allowing developers to manage user interactions with callback functions, ensuring the interface remains responsive. It also supports multi-threading, enabling background tasks like network requests or file operations without interrupting the GUI's performance. Furthermore, Tkinter provides a wide array of input widgets designed to enhance the interactivity and user-friendliness of applications. These widgets accommodate various data input requirements, offering users intuitive controls for effective interaction with the application's features.

- **Textboxes**: Textboxes, also known as entry widgets in Tkinter, allow users to input textual data. They provide a single-line input field where users can type alphanumeric characters, numbers, or symbols. Textboxes are versatile and can be used for various purposes, such as entering text-based search queries, inputting user credentials, or providing feedback.

- **Checkboxes**: Checkboxes are a simple yet effective way to allow users to make binary choices. They consist of a small box that can either be checked or unchecked, indicating the selection status of an option. Checkboxes are commonly used to enable or disable specific features or select multiple items from a list.

- **Comboboxes**: Comboboxes, or dropdown menus, combine the functionality of textboxes and list-boxes, offering users a selection of predefined options in a dropdown list format. Users can either select an option from the list or manually input text if the desired option is not available. Comboboxes are ideal for scenarios where users need to choose from a predefined set of options.

These input widgets can be easily integrated into Tkinter applications using the appropriate widget classes provided by the library. Overall, Tkinter's rich collection of input widgets empowers developers to create highly interactive and user-friendly interfaces that enhance the overall user experience of their applications. On the back-end interface, communication with the board is done via API calls. This API call is structured to send a command to its respective setting on the TOM board's sensors.

5 Investigating Motion Artifact Correction Algorithm

To address the challenge of motion artifacts and enhance the accuracy and reliability of our sensor, our team worked on developing a robust correction algorithm. By analyzing the data collected over previous years, we aimed to quantify and correct the distortions caused by movement. Recognizing the need for continuous improvement, we also continued to collect data to further refine our approach.

This section outlines our systematic approach to improving sensor performance through algorithmic corrections. We began by exploring a variety of algorithms known for effectively managing motion artifacts. Our goal was to run tests on human subjects to collect data and analyze the measurements under various conditions. Through iterative testing and evaluation, we aimed to identify and refine the most effective algorithmic solutions that consistently enhance the accuracy of our sensor readings. The development and ongoing refinement of this correction algorithm are critical to ensuring that our sensor functions reliably, even in dynamic environments.

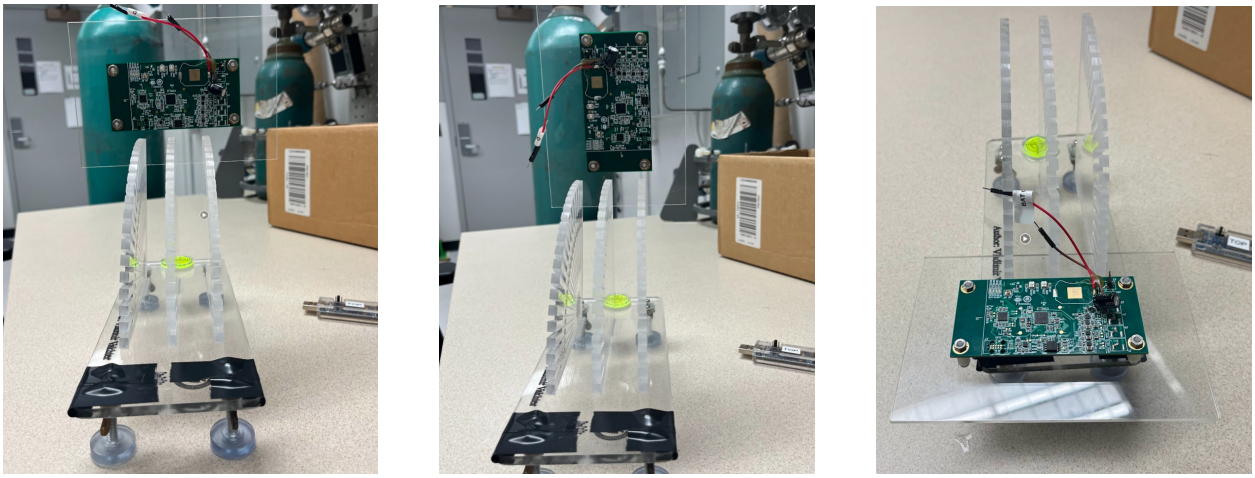
5.1 Methods

In this section, we outline our methodology for creating the motion artifact correction algorithm. This phase of the project involves a sequence of tasks leading up to the selection of the appropriate correction algorithm. These tasks include accelerometer calibration, data acquisition, and algorithmic research. Once these tasks were performed, based on the research conducted, our team chose to implement a Wiener Filter for motion artifact correction, as discussed in the sections below.

5.1.1 Accelerometer Calibration

Achieving a successful calibration relies on the proper setup and orientation of the PCB. This process requires positioning the device accurately and comprehending its mechanisms to guarantee optimal performance throughout the calibration process. Initially to ensure the PCB is correctly aligned, it should be placed on a stable, level surface with its positive x-axis facing upwards, as shown in Fig 4a. Additionally, it is essential to reset the PCB before starting the measurements to ensure data consistency and accuracy.

Once the PCB is set up, start the calibration process for each axis. For the x-axis calibration, the accelerometer's output data with the +x axis pointing upwards is collected, as set up initially. Then flip the PCB so the +x axis points downwards (-x axis) and gather the data once more. For the +y axis calibration, a similar approach is followed: begin by positioning the +y axis upwards (shown in Fig. 4b), gather the



(a) (b) (c)

Figure 4: Accelerometer calibration positions: (a) +x towards gravity vector, (b) +y towards gravity vector, and (c) +z towards gravity vector.

data, then invert the PCB to have the +y axis pointing downwards (-y axis) and record the data again. Lastly, for the +z axis, the same method is adopted: start with the +z axis pointing up (shown in Fig. 4c), collect the required data, then turn the PCB to have the +z axis facing down (for -z axis) and acquire the final set of data. This systematic approach ensures that we capture accurate readings for all axes of the accelerometer. For our accelerometer, we calibrated using three different settings: 2g, 4g, and 8g.

5.1.2 Data Collection

The initial dataset was collected using a human subject. Prior to starting data collection, the sensor's settings were configured to record a total of 912 measurements, correlating with the duration of the test at 76 minutes. Measurements were taken at five-second intervals. The total number of measurements was calculated using the following equation

$$N_{total} = \frac{76\ mins \times 60\ secs/min}{5\ secs} = 912\ measurements \tag{4}$$

The initial 40 minutes of the test, shown in Fig. 5, served as a stabilization phase, during which 480 measurements were recorded. After this phase, three specific movements—Medial Rotation, Flexion, and Lateral Rotation—were each monitored for two minutes. Each motion sequence was followed by a 10-minute stabilization period. To monitor the progression of the test, the team employed a 76-minute timer, carefully observing each scheduled milestone for motion and stabilization periods. The sensor board was powered by a supply set at 3V and 250mA, conforming to the specifications outlined in the reference [36].

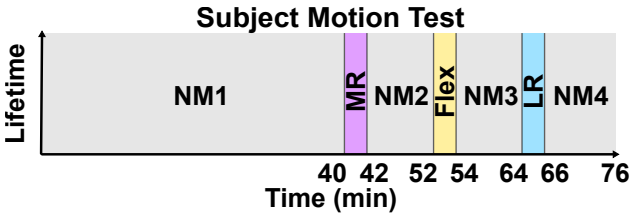


Figure 5: Protocol designed for human subject motion testing (motion types are denoted as NM - No Motion, MR - Medial Rotation, Flex - Flexion, LR - Lateral Rotation).

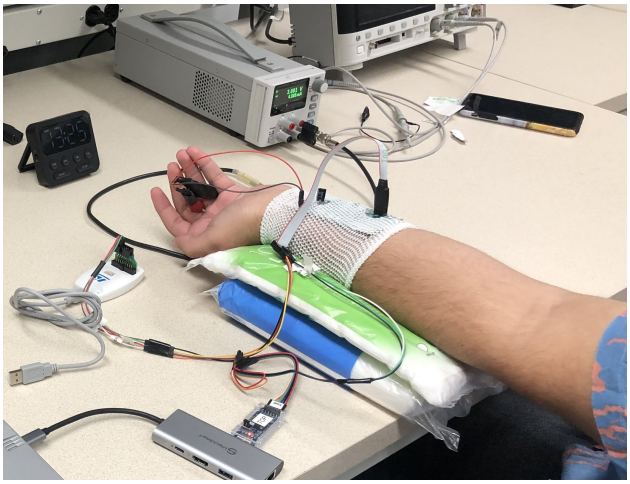


Figure 6: Accelerometer calibration set up.

For the tests, we first sanitized the subject's forearm, after which they wore a sleeve and positioned the PCB board beneath it to ensure proper contact between the sensor and their skin. We then established a connection between the PCB and the computer through the serial communications port, which enabled us to upload the firmware to the board. Data collection started after resetting the PCB and initiating the measurement process. To enhance our dataset, we have devised a detailed plan for data collection that includes varying the initial conditions described in section 6. This plan will involve integrating higher intensity, and quicker motion ranges to investigate how exercise intensity affects oxygen levels.

5.1.3 Motion Artifact Correction Algorithm Research

After collecting data from the $PtcO_2$ sensor, we analyzed the impact of motion artifacts on the data. Our research then shifted to exploring motion artifact correction algorithms applicable to the lifetime measurement raw data. Since the PCB is equipped with an accelerometer to help detect motion artifacts, we focused on algorithms that integrate data from both the sensor and the tri-axis accelerometer. We assessed multiple motion artifact correction algorithms by comparing their average absolute error (AAE) using the IEEE Signal Processing Competition 2015 dataset—a public dataset including signals from two

PPG sensors and a three-axis accelerometer, recorded while human subjects engaged in various activities such as arm exercises and walking [37]. Additionally, we evaluated which methods are most effective for real-time denoising.

5.1.4 Wiener Filter

The Wiener filter, a staple in the domain of optimal filtering, is utilized for signal estimation. When it comes to real-time applications, the challenges posed by non-stationary signals demand a more adaptive approach. The windowed Wiener filter provides a computationally balanced and effective solution for signal estimation by removing estimated or known noise from an observed signal. A Windowed Wiener filter operates by applying the classic Wiener filter within a fixed-sized moving window on the signal. This method is inherently adaptive since the filter is recalculated at every step, based on the latest set of measurements within the current window. This continuous updating allows for the efficient tackling of non-stationary noises and interference.

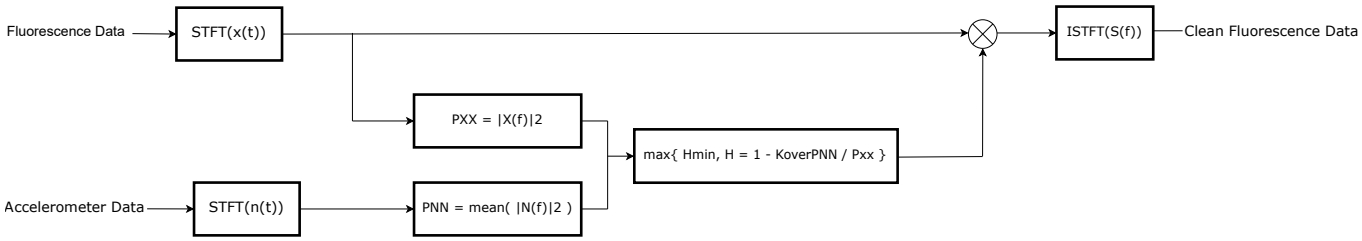


Figure 7: Data flow for classic wiener filter implementation.

In this case, the output of the ADXL367 was used to estimate the noise. The Wiener filter implementation selected involves taking the observed fluorescence and the accelerometer readings into the frequency domain. The frequency spectra are then used to calculate the PSD of the observed signal and estimated noise. The PSD is used to construct the adaptive filter. As shown in Fig. 7 below, the short-time Fourier transform can be used to take the signals from the time domain to the frequency domain. Fig. 7 also shows the use of a minimum filter transfer function, which can help prevent a negative filter. Note that there is also a scale factor (K_{OVER}), which can be used to further scale the PSD of the noise. The filter (H) is applied to the frequency spectra of the observed signal, yielding the estimated clean signal. The estimate of the clean fluorescence signal is brought back into the time domain. This Wiener filter is then repeatedly applied to a series of windows within the signal. Selecting a larger window size results in a smoothing effect, while a smaller window size typically will sharpen the observed signal. Furthermore, the Wiener filter can be applied to a real-time signal by having a rolling window that incorporates a set number of the most recent samples [38]. Note that Fig. 7 visualizes the classic Wiener filter and would need to be iterated upon to

implement the Windowed Wiener filter.

5.2 Results

In this section, we discuss the results of our accelerometer calibration, data collection, and correction algorithm research to improve the accuracy and reliability of our sensor.

5.2.1 Accelerometer Calibration Results

Based on the measurements collected during the accelerometer calibration procedure, we get the $A + 1g$ and $A - 1g$ values for each axis using the Equations 5 and 6. Then, we can calculate the offset and gain for the accelerometer and use it to account for the errors and calculate the actual measurements using equation 7. We also validated our results using Equation 7 to make sure that after applying the offset and gain, we get the expected results.

$$Gain = 0.5 \times \frac{A_{1g} - A_{-1g}}{1g} \tag{5}$$

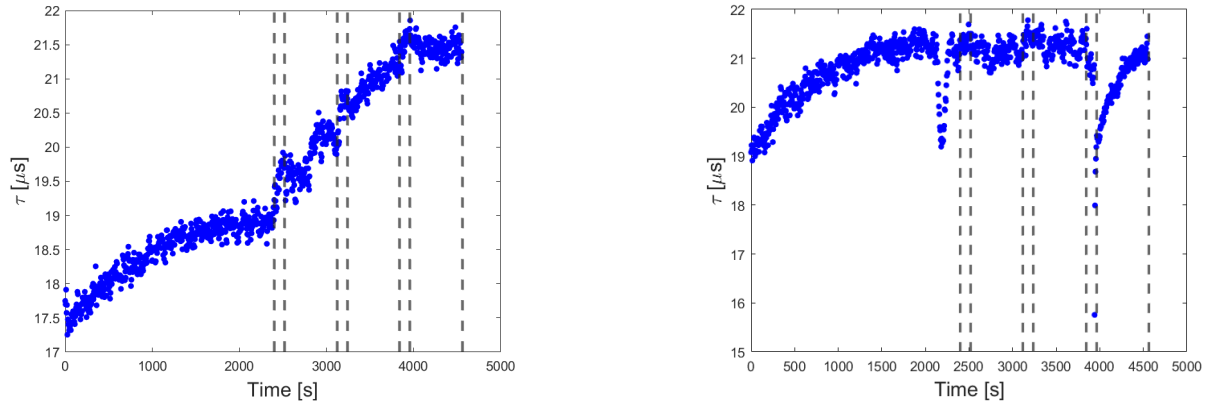
$$A_{OFFs}[g] = 0.5 \times (A_{1g} + A_{-1g}) \tag{6}$$

$$A_{ACTUAL}[g] = \frac{A_{OUT} - A_{OFF}}{Gain} \tag{7}$$

The process of calculating the offset and gain was replicated for all three accelerometer measurement ranges (2g, 4g, 8g). This setting dictates the range of acceleration values in terms of g-force, but higher measurement range results in lower resolution, so typically the range is chosen based on the highest expected acceleration values that need to be measured [39]. For example, we expect the relevant human range to be up to 5g for everyday life, so the measurement range of the accelerometer was set to 8g for the remaining tests, which comes at the cost of losing some acceleration resolution when compared to using 4g or 2g. The table below summarizes the gain and offset for each setting.

Table 1: Gain and Offset for Accelerometer Settings

Setting	Axis	Offset	Gain
2g	x-axis	0.006701	0.975597
	y-axis	0.007600	0.988427
	z-axis	0.0012258	0.9964363
4g	x-axis	0.1691645	0.9834425
	y-axis	-0.0454374	0.9934126
	z-axis	-0.1338606	1.0033883
8g	x-axis	0.1696548	0.9809908
	y-axis	-0.0456008	0.9916147
	z-axis	-0.1374564	1.0001138



(a) (b)

Figure 8: Fluorescence lifetime time constant measurements: (a) Data of test subject A at the 8g accelerometer setting and (b) data of test subject B at the 2g accelerometer setting.

5.2.2 Data Collection Results

When the program finishes running, it stores the data log as a text file in the directory, and then moves any older measurements to the previous runs folder and logs its date. This data then can be transferred into MATLAB to extract the lifetime and accelerometer data and apply any post-processing scripts. The graph in Fig. 8b shows the timeline data for our first subject at the 8G accelerometer setting. It was observed that the lifetime does not stabilize and continues to climb. To mitigate this issue, it was decided to do more testing and modify our method to increase the extra stabilization periods so that the device can be fully stabilized for the action phases. Then using our calibration data, we can apply the offset and gain values to get a more accurate reading.

After testing our subjects, it is evident that we collected data samples that looked stable and provided a more reliable reading from subject B. As with subject A, these measurements on subject B were collected into a text log and a MATLAB script to extract the accelerometer and lifetime data from the ADC measurements. None of our subjects have produced consistent results at the 8g accelerometer setting, which is the desired setting. To adjust for this issue, we shifted our focus to investigating varying stabilization periods before recording the data as well as using a gas chamber for testing rather than a human. From our initial findings, we found that the sensor stabilizes at a lifetime between 22-20 μs on a human.

5.2.3 Motion Artifact Research Results

To evaluate the performance of motion artifact correction algorithms, we generated a chart to compare algorithms that contained desirable attributes. According to [37], the most desirable attributes of these algorithms for a PPG sensor are using singular value decomposition (SVD), frequency domain approaches, and adaptive filtering. Singular value decomposition can isolate the unwanted variations in a signal by decomposing sensor data into components, allowing for the retention of only the most significant features. Adaptive filtering, however, is unsuitable for real-time denoising and thus cannot be employed in our real-time algorithm. In addition, adaptive filtering has high complexity, which means it uses a high amount of power. Retaining the most dominant features of the data, which are typically the signal components, SVD ensures that the essential characteristics of the signal remain intact after denoising, making it an ideal method. Frequency domain approaches are useful for denoising since they provide valuable insights into the behavior of complex signals.

Unfavorable algorithms use singular spectrum analysis (SSA), active noise cancellation (ANC), and normalized least mean square (NLMS) for denoising. SSA only partially removes the motion artifacts from a raw signal and would require another method in conjunction. ANC and NLMS are types of adaptive filtering and, therefore, are not suitable for our implementation. Also, NLMS is a great filtering technique for weaker motion artifacts, but we expect to have a range of weak to strong motion artifacts so it is not ideal for our use case.

We examined each algorithm discussed in [37] and included those that possess desirable attributes in our comparison chart. Table 2 represents the comparison chart that contains the optimal choices of motion artifact correction algorithms based on our research. From this chart, we were able to use the AAE to narrow down the top algorithms. After looking more into the eight-layer model, we learned that it uses a signal from a gyroscope so it was easily ruled out. The MODTRAP algorithm uses EEMD, which was found to be one of the best motion artifact correction algorithms, but its complexity is too high and, therefore, uses too

much power.

Table 2: Motion Artifact Correction Algorithm Comparison Chart

Algorithm	**Year**	**Pre-processing**	**Denoising technique**	**AAE**
SPAMA [40]	2016	Bandpass	Spectral filtering	0.89 ± 0.6
WFPV [9]	2017	4th order butterworth	Wiener filter	1.02
FSM based [41]	2018	4th order butterworth	Wiener filter	0.79 ± 0.6
Cascade and Parallel combo [42]	2018	Bandpass	Cascade and parallel connection of adaptive filters	1.12 ± 2.30
MODTRAP [43]	2020	LPF	EEMD and neural network model	0.57
Eight-layer model [44]	2020	Butterworth BPF	Convolution and max pooling layer	0.76
Precision HR Estimation [45]	2023	HR estimation based on peaks/valleys	Hankel matrix and SVD	1.86
SVD [46]	2016	Bandpass	SVD	1.25 ± 0.6

We looked into a Wiener filter after narrowing down our options even further. A Wiener filter takes into account the frequency characteristics of both the signal and noise which is necessary since the noise and signal components have different spectral profiles. The goal of a Wiener filter is to minimize the mean square error between the estimated signal and the actual signal which will effectively reduce noise but also keep essential characteristics of the lifetime signal [9]. Lastly, a Wiener filter is suitable for real-time applications and has a low computational complexity. Therefore, employing a Wiener filter for noise reduction proved to be our most effective approach.

5.3 Discussion

The development of a simple motion artifact correction algorithm represents significant strides toward enhancing the accuracy and reliability of transcutaneous oxygen monitoring in dynamic environments. Through accelerometer calibration, data collection, algorithmic research, and the development of a Wiener Filter, this project has laid the groundwork for addressing one of the most pervasive challenges in wearable medical monitoring: motion-induced artifacts.

The accelerometer calibration and subsequent data collection phases have provided invaluable insights into the behavior of the sensor under motion. However, the results of the motion tests on human subjects were not ideal, and we needed to further explore stabilization periods, different speeds and lengths of motion, and different oxygen pressure values. Thus, we shifted our focus to testing motion artifacts and

the effects they have on lifetime values in the gas chamber, which is explained in Section 6.

From our initial research and data collection, the Wiener Filter emerged as a suitable solution. It balances the demands for real-time signal processing with the need for minimal computational complexity. This filter's ability to adapt to the signal's changing characteristics makes it particularly well-suited for the task, enabling more accurate transcutaneous oxygen measurements by effectively reducing the noise introduced by the wearer's movements. The preliminary results, as outlined in Section 5.2, have demonstrated the Wiener Filter's potential to enhance sensor data quality, marking a promising advance in the quest for reliable wearable health monitoring. However, more testing still needs to be performed on the sensor to understand the effects of motion artifacts before it is fully implemented.

6 Quantifying the Effects of Motion Artifact

In dynamic environments outside the clinical setting, ensuring the reliability and accuracy of sensor readings becomes vital. Motion artifacts, induced by any form of movement during the monitoring process, can significantly compromise the quality of sensor signals. To address this critical issue, it is essential to thoroughly assess the impact of these artifacts on the performance of our sensor. In this section, we explore how motion artifacts affect sensor functionality through a series of experiments.

We have structured our investigation into three distinct tests: the Single Motion Test, the Dual Motion Test, and the Extended Motion Test. Each test is designed to simulate different real-world scenarios where motion could influence the sensor's readings. By analyzing the outcomes of these tests, we aim to understand the conditions under which the sensor maintains accuracy and identify scenarios where corrective measures may be necessary. The findings from these experiments enhance our understanding of the sensor's robustness and guide the development of strategies to mitigate the effects of motion artifacts in non-clinical environments.

6.1 Methods

To accurately quantify the impact of motion on measurement accuracy, a comprehensive testing methodology has been developed to examine variations in lifetime values under different scenarios. Our approach includes three distinct test protocols, illustrated in Figure 9, each designed to understand how motion artifacts can affect lifetime values under various conditions. Since this sensor is to be part of a wearable device, it is imperative to test various motion scenarios. The device can be used during different motion intensities as well as varying lengths of time. In addition, testing motion at various oxygen levels is crucial to analyze the varied changes in lifetime values, as the nature of the exponential decay curve of PO_2 versus lifetime, depicted in Figure 12, implies that the alterations in lifetime values will differ for each level of PO_2 [7, 15].

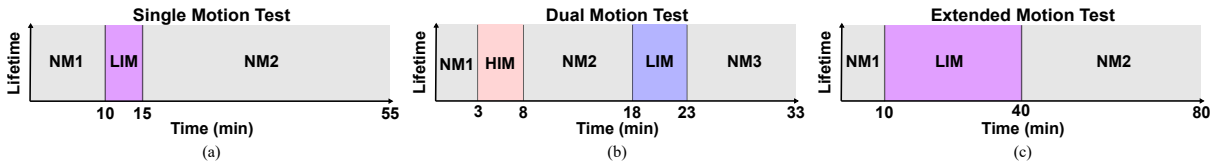
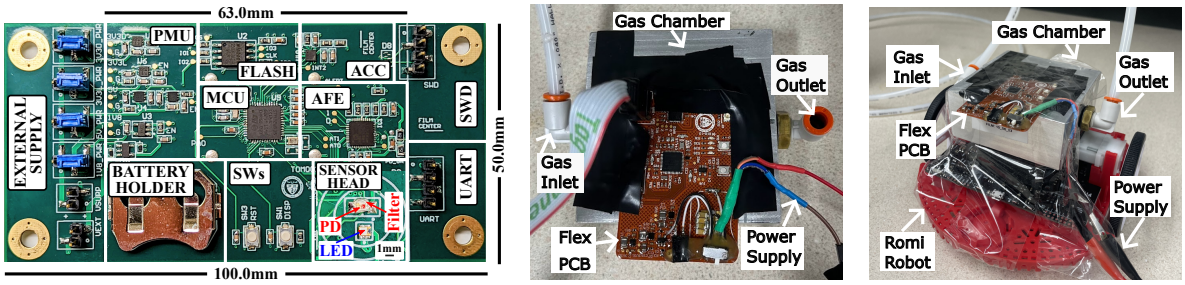


Figure 9: The protocols designed for (a) single motion test, (b) dual motion test, and (c) extended motion test (motion types are denoted as NM - no motion, LIM - low-intensity motion, and HIM - high-intensity motion).

The experiment setup comprises an aluminum gas chamber, within which a luminescent film is placed. Opposite this film, a flexible circuit board equipped with an accelerometer is placed. The flexible circuit board's architecture resembles the PCB with main circuit blocks, illustrated in Fig. 10a. This arrangement aims for efficient data collection by ensuring the sensor directly faces the center of the luminescent film, as illustrated in Fig. 10b. The extended motion experiment is performed by a Romi Robot controlled by a Romi 32U4 control board. The test setup from the previous experiment is used and attached to the robot, as seen in Fig. 10c.



(a) (b) (c)

Figure 10: (a) PCB prototype annotated with main circuit blocks. Motion test setups for (b) single- and dual-motion tests and (c) extended-motion test.

6.1.1 Single Motion Tests at 50 mmHg Oxygen

The first protocol, 'Single Motion,' showcased in Fig. 9a, is designed for a constant PO_2 of 50 mmHg, with a single instance of motion to assess its impact on lifetime values. Selecting 50 mmHg is based on the typical PtcO2 values observed in the absence of heat application in our experiments [47]. Heating is used to increase the oxygen diffusion through the skin in the traditional transcutaneous oxygen monitors [48]. This test allows for the isolation of motion's impact on the sensor's performance, establishing a clear baseline for its responsiveness without the confounding effects of fluctuating oxygen levels.

The initial stability phase begins with a three to ten-minute period of no motion. Initially, the tests were performed with three minutes in the no motion phase. However, to better visualize the baseline at the initial phase, we extended the stabilization period to ten minutes for the remaining tests. This extended duration allows the sensor to settle into the environment, ensuring that a reliable baseline for sensor readings is established without any motion influence. In the following, a five-minute motion period is introduced for assessing the sensor's response to movement, simulating potential real-life disturbances. Concluding the sequence, a 33-to-40-minute stability period follows the motion. This extended final phase is designed to observe the sensor's ability to maintain its baseline readings after experiencing motion, providing

a comprehensive view of its stability over time. The experimental setup in Fig. 10b is utilized for Single Motion tests.

6.1.2 Dual Motion Tests Across Oxygen Variations

In the 'Dual Motion' protocol, motion is induced twice in a single experiment under varying PO_2. During this test, high- and low-intensity motions are tested at four different PO_2 levels. This test protocol provides insight into whether the sensor's performance is affected by consecutive disturbances, which is vital for understanding how it might perform in real-world scenarios where users are not stationary. Furthermore, testing across a range of PO_2 levels with repeated motion events allows for a detailed analysis of how oxygen variability influences the sensor's response to motion.

This experiment investigates the sensor's performance across four different PO_2 levels, from no oxygen present (0 mmHg) to a level resembling that of ambient air ($\sim$ 150 mmHg). To accurately create these specific oxygen conditions, a mixture of high-purity oxygen (OX UHP20, sourced from Airgas) and nitrogen (NI UHP80, also from Airgas) is used. The utilized experimental setup was previously elaborated in [7]. This mixture is carefully adjusted through mass flow controllers to reach the target PO_2 levels: 0 mmHg (0% O_2), 50 mmHg (6.57% O_2), 100 mmHg (13.2% O_2), and 150 mmHg (19.7% O_2). Nitrogen makes up the balance to total 760 mmHg, the standard atmospheric pressure.

To assess the influence of motion artifacts on sensor performance across the various PO_2 levels, the test sequence is designed to mimic real-world conditions where motion intensities can vary, depicted in Fig. 9b for Dual Motion tests. Each test starts with a three-minute initial no motion phase. Subsequently, this phase transitions into five minutes of high-intensity motion, followed by a ten-minute no motion interval to monitor post-motion signal behavior. Following this, a phase of low-intensity motion for five minutes is conducted before entering a concluding stabilization period of ten minutes to again monitor post-motion signal behaviour. The experimental setup in Fig. 10b is utilized for Dual Motion tests.

6.1.3 Extended Motion at 50 mmHg Oxygen

The 'Extended Motion' test is performed at 50 mmHg oxygen pressure but extends the motion duration to evaluate its prolonged effects. This is crucial for continuous monitoring applications, where the sensor must accurately track changes in oxygenation over extended periods, despite ongoing user activity. To observe the effects of a longer period of motion on the sensor, we test movement for 30 minutes, as shown in Fig. 9c. The 30 minutes of movement is performed by a Romi Robot, programmed to move back and

forth 15 cm. Before the movement, there is a ten-minute stabilization period to identify the drop during the motion period. The test setup from the previous experiment is used and attached to the robot as seen in Fig. 10c. The oxygen pressure level used is 50 mmHg. This test helps us to identify the effect of motion artifacts on the signal during extended periods of motion.

6.2 Results

The results of our motion artifact tests provide crucial insights into the operational integrity and reliability of the sensor under various motion conditions. In this section, we present a detailed analysis of the data collected from the Single, Dual, and Extended Motion tests.

Note: All the results presented here are from analysis of raw data without any post processing or correction algorithm.

6.2.1 Single Motion Tests at 50 mmHg

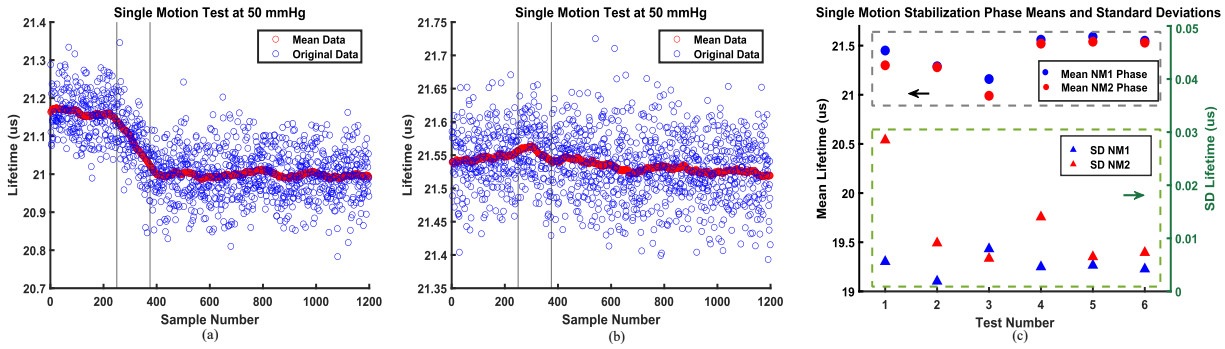


Figure 11: (a) Single motion test with negligible change in lifetime, (b) single motion test with noticeable change in lifetime, (c) means and standard deviations for no motion phases 1 (NM1) and 2 (NM2) during single motion tests.

Upon analysis of the measurement data, we observed two distinct response patterns. Notably, two tests exhibited a clear decrease in lifetime values during the motion phase, as illustrated in Fig. 11a. This contrasted with the remaining tests, which did not present a clear decline during motion periods as seen in Fig. 11b. After the motion, we observed that the signal tends to settle at a new value. The mean and standard deviations during the "No Motion" phase 1 (NM1) and "No Motion" phase 2 (NM2) are shown in Fig. 11c.

Table 3 shows the drops in lifetime values (μs) and the corresponding value in mmHg and percentage. Each 1% of oxygen pressure is equivalent to 7.6 mmHg. The highest drop in PO_2 is 0.11%.

Table 3: Summary of Max Drops in Lifetime Values, Corresponding mmHg, and Percentage in Single Motion Tests

Lifetime Drop (μs)	PO_2 Drop (mmHg)	PO_2 Drop (%)
0.011	0.082	0.011
0.025	0.180	0.024
0.056	0.406	0.053
0.012	0.088	0.012
0.117	0.837	0.110
0.022	0.156	0.021
0.053	0.378	0.051
0.023	0.167	0.022

6.2.2 Method to Quantify the Motion Artifacts

To quantify the effects of motion artifacts during the motion phase, the maximum drops on the mean plot (difference between maximum and minimum values during motion) were calculated. Subsequently, these drops were translated into corresponding PO_2 values in mmHg. This conversion was essential for understanding how much motion artifacts negatively influence the sensor's PO_2 readings. Based on data collected from experiments carried out at four distinct PO_2 values for the dual motion tests, Fig. 12 was created to illustrate the relation between PO_2 and lifetime values.

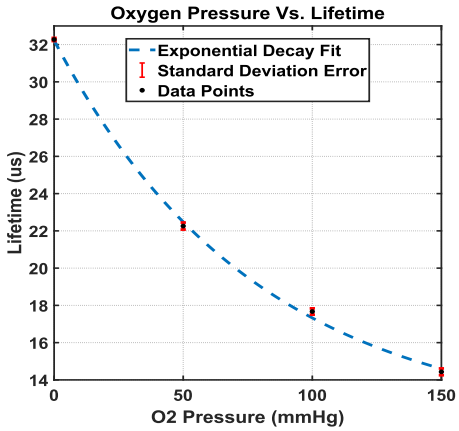


Figure 12: Decay curve used for lifetime to PO_2 conversion.

The black data points signify the average lifetime value corresponding to each PO_2, while the red error bars illustrate the deviation in lifetime value observed during each maximum drop. The data points

were fit to an exponential decay model, shown as a dashed blue line in Fig. 12, and the fit equation is given as follows:

$$\tau = 20.6905 \times e^{-0.0127x} + 11.5368, \tag{8}$$

where the variables x and τ represent PO_2 in mmHg and lifetime in μs, respectively. With this equation, we calculate the derivative at the four different PO_2 values used in the dual motion tests. The derivative is the rate of change in mmHg/μs. This rate is then used in a linear equation as the slope and maximum τ drops serve as the dependent variable. Through this approach, we can determine the maximum PO_2 drop in mmHg and the oxygen percentage from the lifetime values that the prototype measures.

6.2.3 Dual Motion Test Results

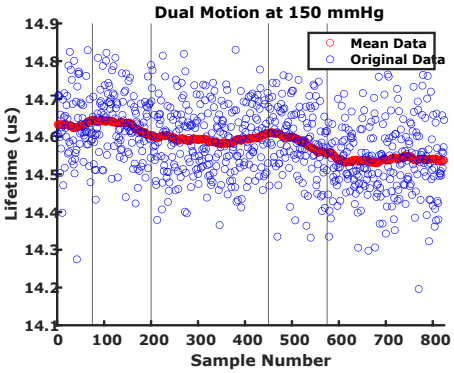


Figure 13: Dual motion test.

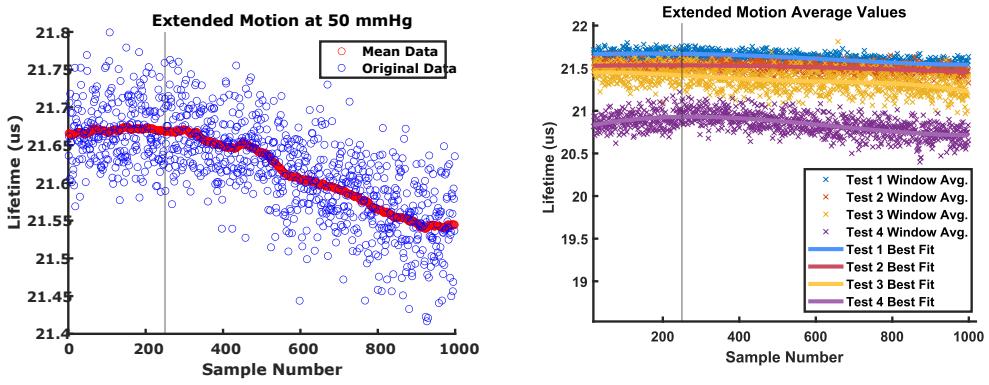
Fig. 13 illustrates an example of the signal output obtained during the dual motion test at a PO_2 of 150 mmHg. To quantitatively evaluate the extent of artifacts during dual motion experiments, we determined the maximum decrease in lifetime values during motion, observed across each oxygen level tested.These maximum drops were converted into corresponding PO_2 in Table 4. It is evident from Table 4 that none of the observed drops exceed 1% PO_2. The most substantial drop is 5.102 mmHg (0.67%), indicating that motion artifacts have a minimal effect on the PO_2 output of the sensor. In addition, the standard deviation of lifetime, represented by the error bars in Fig. 12, is not significant.

Table 4: Summary of Max Drops in Lifetime Values, Corresponding mmHg, and Percentage in Dual Motion Tests

PO_2 (mmHg)	Lifetime Drop (μs)	PO_2 Drop (mmHg)	PO_2 Drop (%)
150	0.165	4.219	0.555
100	0.377	5.102	0.671
50	0.337	2.422	0.319
0	0.363	1.383	0.182

6.2.4 Extended Motion at 50 mmHg

The extended motion tests provide a similar result to the dual motion tests. In Fig. 14a, the plotted data reveals a drop during the motion period of the test that tapers off towards the end of the 30-minute motion period around 900 samples.



(a) (b)

Figure 14: (a) Extended motion test, (b) Average Results of Extended Motion Tests

The results from the extended motion tests are visualized in Fig. 14b. All tests were performed under the same PO_2 conditions with the same sensor film and board on different days. Cross markers represent the moving average measurements, computed with a window length of one hundred. Solid lines depict the polynomial fit with a degree of 4, serving as the line of best fit. Additionally, a vertical line at 250 samples denotes the start of the motion period. From each average test, it is evident that the slope decreases as time increases. The total drops in PO_2 values varied between 2 mmHg to about 6 mmHg with the highest value being 6.054 mmHg (0.797 %). It is evident that the longer motion does not increase the drop in lifetime and the values stop decreasing around 15 minutes.

The Food and Drug Administration (FDA) recommends that the accuracy of $PtcO_2$ should be within 5 mmHg over the range of 0 to 20.9% and 10 mmHg for the higher oxygen range [49]. The presented values in Table 3 are consistently lower than the values recommended by the FDA. The values in Table 4

are also mainly lower than 5 mmHg except one slightly higher than that value. In the extended motion tests, the $PtcO_2$ change reaches a maximum of 6.054 mmHg (0.797 %). These results indicate the robustness of the lifetime-based measurements of luminescence-based transcutaneous oxygen prototype against motion artifacts even with the raw data without post-processing applied.

6.3 Discussion

This section has detailed efforts to quantify and understand the impact of motion artifacts on a transcutaneous oxygen measurement sensor, highlighting the pursuit of a reliable, wearable, noninvasive blood oxygen sensor. Grounded in previous research, the findings contribute significantly to the comprehension of the sensor's behavior in the presence of motion. Through the setup and execution of three experiments—dual motion, single motion, and extended motion—the resilience of this luminescence-based sensor against motion artifacts was demonstrated, showing less than 1% sensor error even when subjected to motion.

Compared to the motion artifacts found in other light-based technologies, such as PPG and fNIRS, the luminescence-based transcutaneous oxygen monitor with lifetime measurement technique demonstrates less vulnerability against motion artifacts. At most, the motion artifacts caused a (PaO_2) error of about 6 mmHg at an extended motion duration. These findings support the potential for this sensor in applications that require reliable, long-term, wearable, continuous, oxygen monitoring. Future motion artifact testing could involve long-term human testing to better quantify long-term drift and deviation and the development of motion artifact correction to further improve sensor accuracy.

7 BLE Development

The application of BLE within sensor development opens a new frontier of possibilities. Sensors, such as those monitoring environmental conditions, health parameters, or equipment status, benefit immensely from BLE's capabilities. The initiative to adopt BLE for the transcutaneous oxygen monitor was primarily motivated by the challenges encountered with wired connections, which often became disconnected during motion artifact testing. Through the adoption of BLE technology, the transcutaneous oxygen monitor can achieve operational longevity while enabling real-time data transmission and facilitating seamless connectivity with other devices.

In Section 7.1, we will explore the methods and processes employed in developing a sensor-specific BLE application for the transcutaneous oxygen monitor. Next, in Section 7.2, we will look at the current implementation of the BLE application as well as the challenges faced during development. Finally, Section 7.3 will discuss the outcomes of this application and the future directions for this project, emphasizing the untapped potential and areas for further exploration.

7.1 Methods

This methods section will cover the flow of BLE development and the resources and knowledge used to support that in Section 7.1.1. Then, we will dive into step-by-step instructions on the implementation of a custom BLE server in Section 7.1.2.

7.1.1 BLE Development

SoC: STM32WBx5 The integration of BLE technology into our oxygen sensing device started with a foundational emphasis on understanding our selected system on chip (SoC), specifically an STM32-based microcontroller unit (MCU) renowned for its support of wireless communication protocols, including BLE. The SoCs used for development were part of the STM32WBx5 series, which have comprehensive support for BLE and also align with our project's requirements for efficient, reliable wireless communication. This series was used for previous sensor development [17, 36], but the sensor had not yet utilized the wireless capability of the SoC.

The process of familiarization began with a deep dive into STM's documentation related to the SoC by utilizing datasheets [33], application notes [33], wiki articles, and video content [50, 51]. Through this growth in understanding, we learned that the STM32WB MCUs are a very capable solution towards meeting

the demanding requirements of modern IoT applications. Below are the standout features and benefits that made the STM32WBx5 series ideal, compared to competitors such as the ESP32 and NRF52 [52, 53], for the transcutaneous oxygen monitor [33]:

- **Power Efficiency**: Power consumption is a critical factor in the design of portable medical devices. The STM32WBx5 series excels in this area with its energy-efficient design and various power-saving modes, allowing for extended battery life. This ensures that the oxygen monitor can operate for longer periods without frequent recharging, making it more convenient and reliable for users.
- **Security Features**: Security is paramount in healthcare applications to protect sensitive patient data. The STM32WBx5 series comes equipped with advanced security features ensuring that all data transmitted and received over BLE is securely encrypted.
- **Flexibility and Scalability**: The modular architecture of the STM32WBx5 series allows for significant flexibility and scalability in design. This enables the integration of additional sensors or communication protocols, should the need arise, making the platform suitable for future enhancements or iterations of the oxygen monitor.
- **Integration of Multiple Wireless Protocols**: Besides BLE, the STM32WBx5 series supports other wireless protocols, offering the potential for multi-protocol wireless applications. This could be leveraged in future iterations of the oxygen monitor.
- **Robust Ecosystem and Development Tools**: STM provides an extensive ecosystem of development tools, software libraries, and support resources for the STM32WBx5 series.

BLE Architecture The next step in gaining the foundational understanding necessary for this project involved understanding Bluetooth and, more specifically, BLE. Similar to before, this involved reviewing articles [32], wiki articles [33, 34], and videos [50, 51]. This learning led to the design of a generic attribute (GATT) profile, a key component of our BLE implementation. GATT plays a fundamental role in BLE communication, acting as the backbone through which data is structured, stored, and exchanged between BLE devices. GATT operates on the principle of a server-client model, where the GATT server stores data and makes it accessible to the client upon request using a BLE protocol stack. This model is essential for BLE devices, as it dictates how data is formatted and transferred over BLE's low-energy protocol through the use of services and characteristics. In the context of our project, the GATT server was designed to cater to the specific data requirements of the oxygen-sensing device. We designed a custom profile comprising two primary services: sensor configuration and sensor data transmission.

The profile implementation is shown in Table 5, showcasing the service used for data transmission and an example service for sensor configuration. The sensor configuration service has the potential to enable the remote setup and calibration of the oxygen sensor, allowing users to tailor the device's operation to their specific needs. Currently the sensor configuration service only consists of example characteristics for demonstrating the functionality of notifying, reading, and writing with characteristics in conjunction with hardware, such as switch buttons and LEDs. This service intends to create a foundation via examples for when the BLE server integration within the sensor firmware has been resolved and we are ready to incorporate useful characteristics for sensor configuration. The sensor data transmission service, which will likely is responsible for the real-time streaming of collected data, which includes critical parameters such as temperature, accelerometer readings, and tau values indicating $PtcO_2$ levels. Specific details of the services and characteristics can be seen in the table, such as the amount of data, in bytes, each characteristic is responsible for. More information can be accessed via the ICAS lab on their GitHub.

Table 5: BLE Profile Structure

Service Long Name	**SensorConfiguration**		**SensorData**		
Service Short Name	SConfig		SData		
UUID Type	128 bits		128 bits		
Characteristic Long Name	My_LED_Char	My_Switch_Char	TOM_Temp	TOM_Accel	TOM_Tau
Characteristic Short Name	LED_C	SWITCH_C	TOM_T	TOM_A	TOM_TAU
UUID Type	128 bits	128 bits	128 bits	128 bits	128 bits
Char Properties	Read + Write w/o response	Notify	Notify	Notify	Notify
Char Value Length	2	2	2	6	200

STM32 Development Suite The implementation of BLE began with an in-depth familiarization with STM32 hardware and its development suite [54]. This initial phase was crucial for establishing a solid foundation of understanding, given the breadth and depth of the tools available for hardware development from STM32 and involved running example codes from the STM32CubeWB GitHub Repository on the P-Nucleo Development Board [55]. The next phase involved an immersive learning experience on BLE development, especially concerning the STM32 ecosystem. This consisted of leveraging wiki articles/guides and video tutorials [50,51,56], to acquire the necessary skills and knowledge for custom BLE implementation. This theoretical groundwork paved the way for the practical application phase, developing a custom BLE server using STM32's P-Nucleo-WB55 development board, using the STM32WB55. The next step, which is

currently in progress, involves integrating the BLE server onto the existing flexible PCB sensor hardware, which necessitates overcoming a few significant differences in features. The hurdles include a difference in pinout, a different MCU (the flex PCB incorporates the STM32WB35), and the lack of a functional low-speed external (LSE) clock.

7.1.2 BLE Guides

The following guides walk through BLE related procedures and are available on the ICAS Lab GitHub repository (https://github.com/icaslab/tom-mcu-wireless-fw).

Custom BLE Server Procedure

This procedure walks through the creation of a BLE server. You can refer to the guides (later in this section) for creating "notify" and "write" characteristics for more detail on that.

Prerequisites:

- Install STM32CubeIDE (optionally you can install STM32CubeMX too, but most features are available in STM32CubeIDE).
- Update Firmware Upgrade Service (FUS) and flash wireless stack. BLE Hardware Setup Instructions. If you are using the P-Nucleo it is recommended to flash this P2P Server Application to confirm BLE is functional (will not work on flex PCB). You will need STM32CubeProgrammer to update FUS and flash wireless stack.
- Get the BLE development app on the client device. LightBlue for IOS is a good option.
- Get PuTTY or some other serial console.

1. Define the desired services and characteristics of your application

For our application, we use the setup shown in Table 5.

Note: Recommended to use a UUID generator to create custom 128-bit UUIDs, but, if preferred, can use assigned 16-bit UUIDs by referring to Assigned Numbers.

2. Initialize CubeMX, Pinout and Configuration, Clock Configuration, and Project Manager

You can do this by following Cection 4.1-4.4 of STM32CubeMX Application Conception.

Note: For step 4.4 select STM32CubeIDE as Toolchain/IDE.

3. Enable Traces!

To get feedback on BLE events and activity, you will want to enable traces. You can follow section 6 of STM32CubeMX Application Conception and don't forget to add APPD_Init(); to app_entry.c.

4. Setup custom BLE application in STM32_WPAN.

Refer to Section 5 of STM32CubeMX Application Conception for example. This is where you configure your services and characteristics.

a. Setup applications and services

BLE Wireless Stack	
BLE Wireless Stack	Full
BLE Application Type	
BLE Application Type	Server profile
Server Mode	
BT SIG Beacon	Disabled
BT SIG Blood Pressure Sensor	Disabled
BT SIG Health Thermometer Sensor	Disabled
BT SIG Heart Rate Sensor	Disabled
Custom P2P Server	Disabled
Custom Template	Enabled
BLE Services Configuration	
The device needs to support the Peripheral Role	1
The device needs to support the Central Role	0
BLE_CFG_SVC_MAX_NBR_CB	7
BLE_CFG_CLT_MAX_NBR_CB	0

Figure 15: Setup application in BLE Applications and Services tab.

b. Setup advertising configuration

Advertising configuration	
Advertising Type	Undirected scannable and connectable(0x00)
CFG_IDENTITY_ADDRESS	GAP_PUBLIC_ADDR
CFG_PRIVACY	Disabled
Advertising Filter	No white list(0x00)
Peripheral: Advertise and connectable	No (0x00)
Broadcaster: Advertise and non-connectable	No (0x00)
Central: Scan and connect	No (0x00)
Observer: Scan	No (0x00)
CFG_GAP_DEVICE_NAME	TOM_BLE
CFG_GAP_DEVICE_NAME_LENGTH	7

(a)

Advertising elements	
ad_data[] length	13
Include AD_TYPE_TX_POWER_LEVEL element	Yes
AD_TYPE_TX_POWER_LEVEL_LENGTH	2
AD_TYPE_TX_POWER_LEVEL	(0x18) /* -0.15dBm */
Include AD_TYPE_COMPLETE_LOCAL_NAME element	Yes
AD_TYPE_COMPLETE_LOCAL_NAME_LENGTH	4
AD_TYPE_COMPLETE_LOCAL_NAME	TOM
Include AD_TYPE_SHORTENED_LOCAL_NAME element	No
Include AD_TYPE_APPEARANCE element	No
Include AD_TYPE_ADVERTISING_INTERVAL element	No
Include AD_TYPE_LE_ROLE element	No
Include AD_TYPE_16_BIT_SERV_UUID_CMPLT_LIST element	No
Include AD_TYPE_128_BIT_SERV_UUID_CMPLT_LIST element	No
Include AD_TYPE_SLAVE_CONN_INTERVAL element	No
Include AD_TYPE_URI element	No
Include AD_TYPE_MANUFACTURER_SPECIFIC_DATA element	Yes
AD_TYPE_MANUFACTURER_SPECIFIC_DATA_LENGTH	4
Company identifier	30,00
Number of user defined data item(s)	1
User defined data 1	00
Comment data 1	

(b)

Figure 16: Configure advertising in BLE Advertising tab. Choose CFG_GAP_DEVICE_NAME here. Here we use TOM_BLE.

c. Define services and characteristics

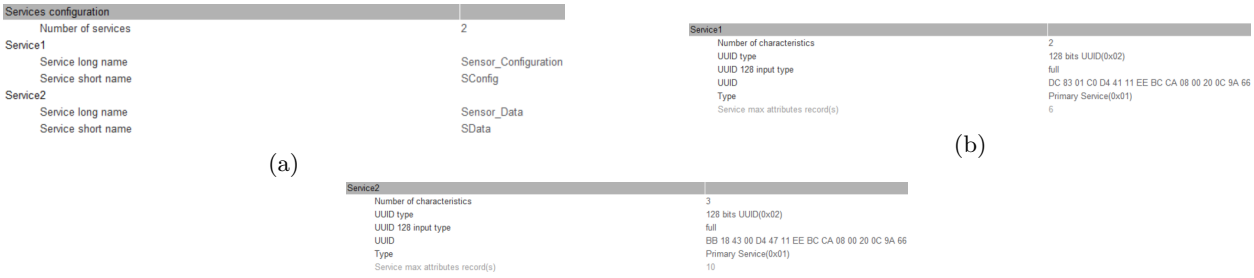


Figure 17: Configure services in BLE GATT and the new services tabs.

d. Define pairing (optional security features)

See Section 5.3.3 of STM32CubeMX Application Conception or BLE Security with STM32WB for more information.

Guide for Characteristic Notification

This guide walks through how to get notify functionality with a characteristic. We will use a button to trigger a notification for this guide, but you can use this as a reference to notify with hardware or software triggers.

Prerequisites:

Make sure you have your notification trigger set up, whether it's a timer, button, or a software sequence.

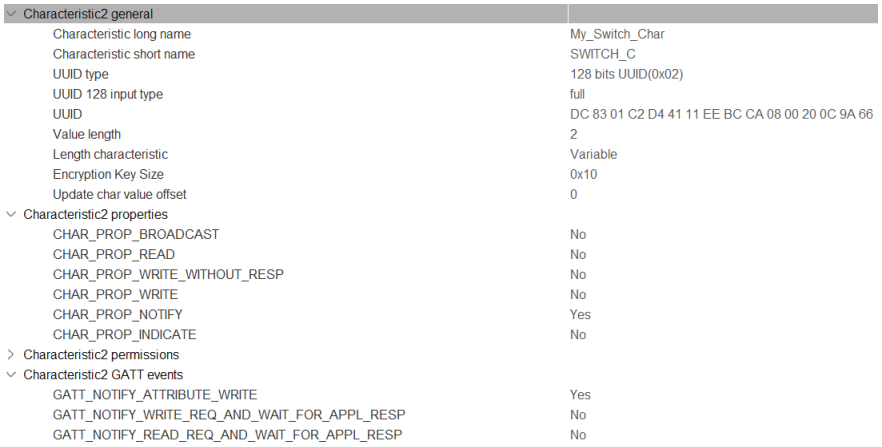
1. Setup characteristic in your .ioc (can use CubeMX or CubeIDE)

Make sure that CHAR_PROP_NOTIFY and GATT_NOTIFY_ATTRIBUTE_WRITE are enabled. Also, make sure your value length is enough to contain your data and note that 'Variable' length characteristic does not mean that you can exceed your value length (Fig. 18).

2. Add task to sequencer

We will name the task after the cause of the notification, which in our case is a button (Figure 19).

- `app_conf.h` - CFG_Task_Id_With_HCI_Cmd_t



Characteristic2 general	
Characteristic long name	My_Switch_Char
Characteristic short name	SWITCH_C
UUID type	128 bits UUID(0x02)
UUID 128 input type	full
UUID	DC 83 01 C2 D4 41 11 EE BC CA 08 00 20 0C 9A 66
Value length	2
Length characteristic	Variable
Encryption Key Size	0x10
Update char value offset	0
Characteristic2 properties	
CHAR_PROP_BROADCAST	No
CHAR_PROP_READ	No
CHAR_PROP_WRITE_WITHOUT_RESP	No
CHAR_PROP_WRITE	No
CHAR_PROP_NOTIFY	Yes
CHAR_PROP_INDICATE	No
Characteristic2 permissions	
Characteristic2 GATT events	
GATT_NOTIFY_ATTRIBUTE_WRITE	Yes
GATT_NOTIFY_WRITE_REQ_AND_WAIT_FOR_APPL_RESP	No
GATT_NOTIFY_READ_REQ_AND_WAIT_FOR_APPL_RESP	No

Figure 18: Setup characteristic in your .ioc.

```
664 typedef enum
665 {
666   CFG_TASK_ADV_CANCEL_ID,
667 #if (L2CAP_REQUEST_NEW_CONN_PARAM != 0 )
668   CFG_TASK_CONN_UPDATE_REG_ID,
669 #endif
670   CFG_TASK_HCI_ASYNCH_EVT_ID,
671   /* USER CODE BEGIN CFG_Task_Id_With_HCI_Cmd_t */
672   CFG_TASK_SW1_BUTTON_PUSHED_ID,
673   CFG_TASK_SW2_BUTTON_T_PUSHED_ID,
674   CFG_TASK_SW2_BUTTON_A_PUSHED_ID,
675   CFG_TASK_SW2_BUTTON_TAU_PUSHED_ID,
676 //  CFG_TASK_TIM2_TRIG_ID,
677   /* USER CODE END CFG_Task_Id_With_HCI_Cmd_t */
678   CFG_LAST_TASK_ID_WITH_HCICMD,
679 } CFG_Task_Id_With_HCI_Cmd_t;
```

Figure 19: Add task to sequencer.

3. Setup event chain

Here, we set up the chain of callbacks to trigger the notification. Since we are using a button, we create a callback for the button (Figure 20a), then create an app action function to step into (this is your bridge from CPU1 to CPU2) (Figure 20b), and finally create the BLE function to set the flag for the event for the sequencer (Figure 20c). Make sure you declare the functions you create in their respective include files!

- `app_entry.c` - Callback Function (In the case of an interrupt-related event. For software-related events, you can put your function wherever else makes sense as long as the code is running on CPU1.)
- `app_ble.c` - Action Function
- `custom_app.c` - Set Event Function

```c
/* USER CODE BEGIN FD_WRAP_FUNCTIONS */
void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin)
{
  switch (GPIO_Pin)
  {
    case SW1_User_Pin:
      APP_BLE_Key_Button1_Action();
      break;
    case SW2_User_Pin:
      APP_BLE_Key_Button2_Action();
      break;
    default:
      break;
  }
  return;
}
```

(a)

```c
/* USER CODE BEGIN FD_SPECIFIC_FUNCTIONS */
void APP_BLE_Key_Button1_Action(void)
{
  P2PS_APP_SW1_Button_Action();
}
```

(b)

```c
/* USER CODE BEGIN FD_LOCAL_FUNCTIONS*/
void P2PS_APP_SW1_Button_Action(void)
{
  UTIL_SEQ_SetTask(1<<CFG_TASK_SW1_BUTTON_PUSHED_ID, CFG_SCH_PRIO_0);

  return;
}
```

(c)

Figure 20: Setup event chain.

4. Setup notification flow

Here, we start by creating our function to send the notification (Fig. 21a). This function relies on the button as a toggle using SW1_Status, so we need to set that up by adding it to Custom_App_Context_t (Fig. 21b). We also need to add the ability for the client device to decide the notification status by adding to Custom_STM_App_Notification (Fig. 21c). Finally, we can initialize the notification and register the notification task with the sequencer in Custom_APP_Init (Fig. 21d).

- `custom_app.c` - All remaining functions in Figure 21, including Custom_App_context_t, Custom_STM_App_Notification(), and Custom_APP_Init().

Guide for Characteristic Writing

This guide walks through how to get "writing" functionality with a characteristic. We toggle an LED state as an example for this guide, but you can use this as a reference to change hardware or software states.

Prerequisites:

Make sure you have your writing destination set up, whether it's an LED or a variable.

1. Setup characteristic in your .ioc (can use CubeMX or CubeIDE)

Make sure that CHAR_PROP_WRITE or CHAR_PROP_WRITE_WITHOUT_RESP is enabled. Also, make

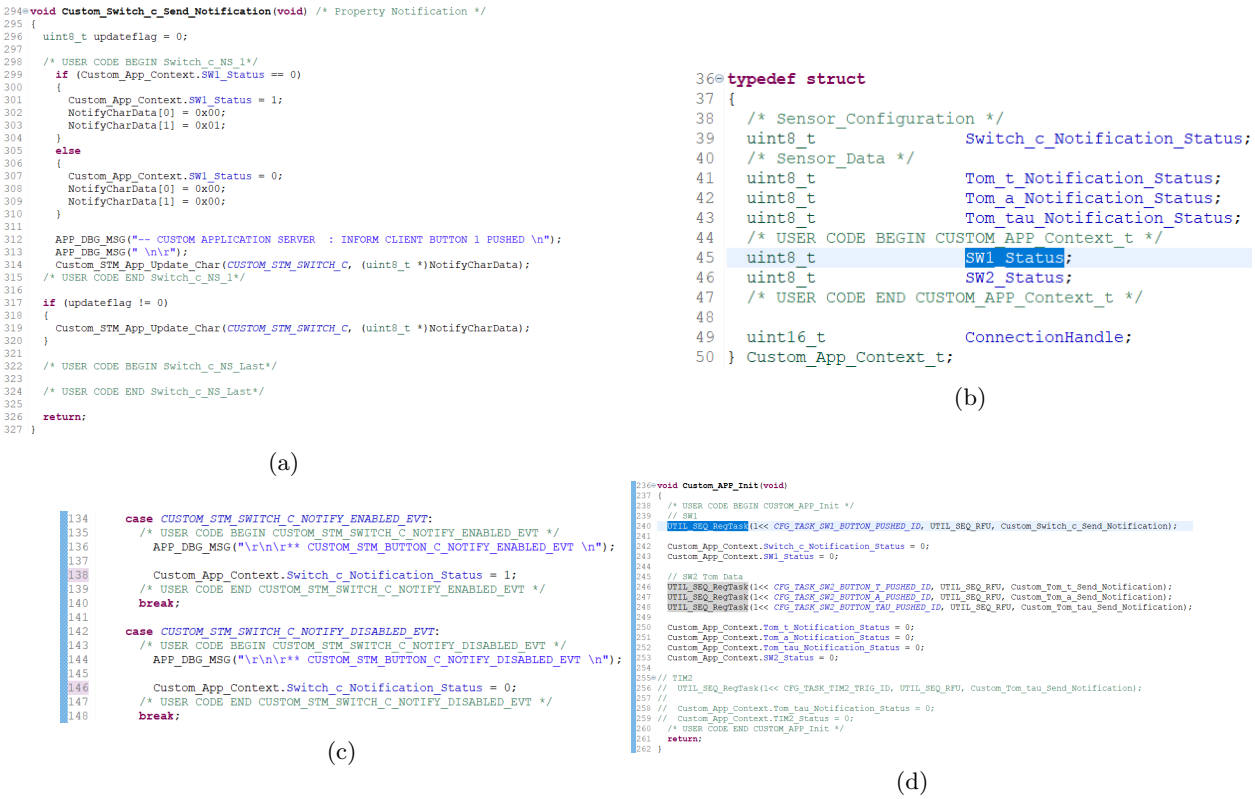


Figure 21: Setup notification flow.

sure your value length is enough to contain your data and note that “Variable” length characteristic does not mean that you can exceed your value length (Figure 18).

Characteristic1 general	
Characteristic long name	My_LED_Char
Characteristic short name	LED_C
UUID type	128 bits UUID(0x02)
UUID 128 input type	full
UUID	DC 83 01 C1 D4 41 11 EE BC CA 08 00 20 0C 9A 66
Value length	2
Length characteristic	Variable
Encryption Key Size	0x10
Characteristic1 properties	
CHAR_PROP_BROADCAST	No
CHAR_PROP_READ	Yes
CHAR_PROP_WRITE_WITHOUT_RESP	Yes
CHAR_PROP_WRITE	No
CHAR_PROP_NOTIFY	No
CHAR_PROP_INDICATE	No
Characteristic1 permissions	
Characteristic1 GATT events	
GATT_NOTIFY_ATTRIBUTE_WRITE	Yes
GATT_NOTIFY_WRITE_REQ_AND_WAIT_FOR_APPL_RESP	No
GATT_NOTIFY_READ_REQ_AND_WAIT_FOR_APPL_RESP	No

Figure 22: Setup characteristic in your .ioc.

2. Setup write management

Add code to the relevant condition in Custom_STM_Event_Handler function to manage the write from a client device to your server. In this case, our condition is CustomLed_C, which matches our characteristic

definition (Figure 23).

- `custom_stm.c` - Custom_STM_Event_Handler()

```
338           else if (attribute_modified->Attr_Handle == (CustomContext.CustomLed_CHdle + CHARACTERISTIC_VALUE_ATTRIBUTE_OFFSET))
339           {
340             return_value = SVCCTL_EvtAckFlowEnable;
341             /* USER CODE BEGIN CUSTOM_STM_Service_1_Char_1_ACI_GATT_ATTRIBUTE_MODIFIED_VSEVT_CODE */
342               Notification.Custom_Evt_Opcode = CUSTOM_STM_LED_C_WRITE_NO_RESP_EVT;
343               Notification.DataTransfered.Length=attribute_modified->Attr_Data_Length;
344               Notification.DataTransfered.pPayload=attribute_modified->Attr_Data;
345               Custom_STM_App_Notification(&Notification);
346             /* USER CODE END CUSTOM_STM_Service_1_Char_1_ACI_GATT_ATTRIBUTE_MODIFIED_VSEVT_CODE */
347           } /* if (attribute_modified->Attr_Handle == (CustomContext.CustomLed_CHdle + CHARACTERISTIC_VALUE_ATTRIBUTE_OFFSET))*/
348           /* USER CODE BEGIN EVT_BLUE_GATT_ATTRIBUTE_MODIFIED_END */
349
350           /* USER CODE END EVT_BLUE_GATT_ATTRIBUTE_MODIFIED_END */
```

Figure 23: Setup write management.

3. Setup write action

Add code to write event in Custom_STM_App_Notification function. This code should handle the desired effect of your write. In this case, the write changes an LED state depending on its value (Figure 24).

- `custom_app.c` - Custom_STM_App_Notification()

```
119   case CUSTOM_STM_LED_C_WRITE_NO_RESP_EVT:
120     /* USER CODE BEGIN CUSTOM_STM_LED_C_WRITE_NO_RESP_EVT */
121       APP_DBG_MSG("\r\n\r** CUSTOM_STM_LED_C_WRITE_NO_RESP_EVT \n");
122       APP_DBG_MSG("\r\n\r** Write Data: 0x%02X %02X \n", pNotification->DataTransfered.pPayload[0], pNotification->DataTransfered.pPayload[1]);
123       if(pNotification->DataTransfered.pPayload[1] == 0x01)
124       {
125         HAL_GPIO_WritePin(Blue_Led_GPIO_Port, Blue_Led_Pin, GPIO_PIN_SET);
126       }
127       if(pNotification->DataTransfered.pPayload[1] == 0x00)
128       {
129         HAL_GPIO_WritePin(Blue_Led_GPIO_Port, Blue_Led_Pin, GPIO_PIN_RESET);
130       }
131     /* USER CODE END CUSTOM_STM_LED_C_WRITE_NO_RESP_EVT */
132     break;
```

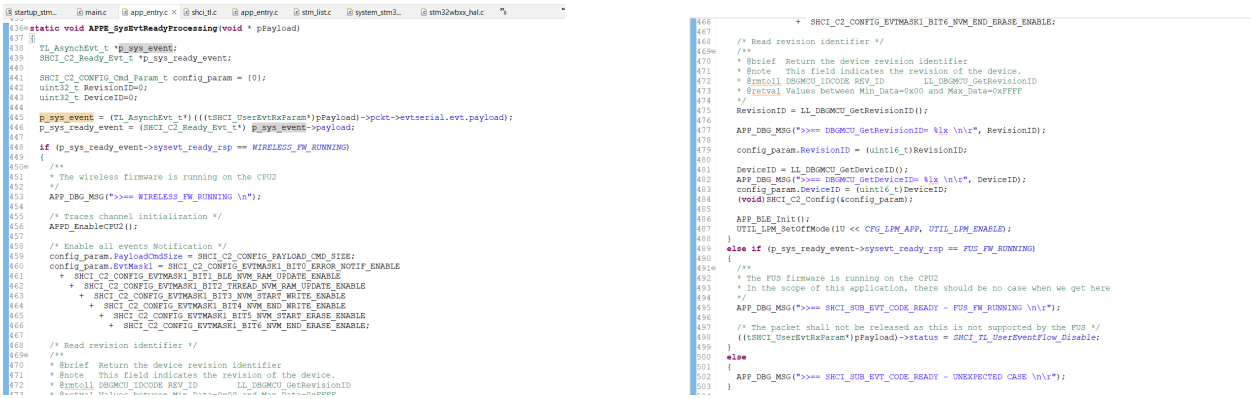
Figure 24: Setup write action.

Helpful STM32 tutorials/guides:

- STM32CubeMX Application Conception
- BLE Security with STM32WB
- How to build wireless applications with STM32WB MCUs
- STM32 Bluetooth Firmware Tutorial (Bring-Up) - Phil's Lab #129

7.2 Results

The implementation of BLE on the flex PCB could not be completed due to various issues with software and hardware. The LSE clock on the flex PCB does not function as necessary, which requires a workaround involving referencing the high-speed external (HSE) clock. In other words, the LSE clock is



(a) (b)

Figure 25: Source of issue for implementing BLE server on flex PCB: (a) ideally sysevt_ready_rsp matches WIRELESS_FW_RUNNING, however, (b) instead sysevt_ready_rsp undesirably matches FUS_FW_RUNNING.

not strictly necessary, but there needs to be a lower speed reference to utilize lower power modes for BLE. Furthermore, an issue relating to the wireless stack not initializing properly prevents CPU2 from running, which is important since CPU2 handles all wireless communication. Unfortunately, this issue is rare and is not well documented.

Other than the implementation of BLE on the flex PCB, the accomplishments of BLE development exist on the TOM Wireless GitHub repository (https://github.com/icaslab/tom-mcu-wireless-fw). The following contributions were made to this repository toward creating a foundation for BLE development on the sensor:

- Branch with most recent flex PCB codes.
- Resources relating to BLE development on the STM32.
- Guides for creating a BLE server from scratch for any application.
- Guides for implementing characteristics for reading, writing, and notification.
- Branch with test BLE server for P-Nucleo.

As mentioned before, the flex PCB codes are in a working state. The origin of the wireless stack issue, depicted in Figure 25 needs to be determined before further progress can be made.

Detailed guides containing helpful resources, many of which were discussed above, are available on the GitHub repository, partially shown in Figure 26. These resources cover general topics useful for understanding BLE, STM, and the MCU. The repository also contains resources useful for creating BLE

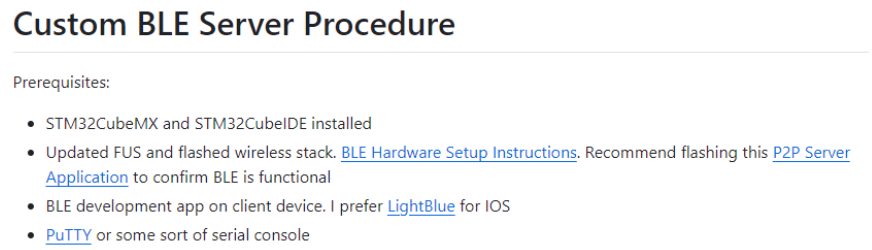


Figure 26: A peek at the documentation available on the GitHub repository.

applications from scratch using the STM development tools. These tools have a steep learning curve, so the guides are meant to provide simple guidance that was honed with trial and error throughout the course of this project.

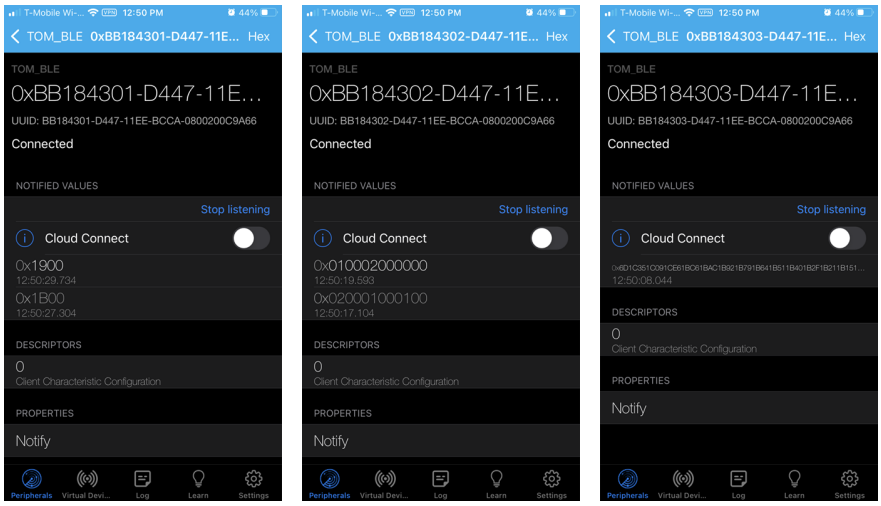


Figure 27: Example BLE data transmitted from P-Nucleo seen on IOS.

Figure 27 shows the example BLE data transmitted from the P-Nucleo development board. From left to right, each screenshot shows temperature data, accelerometer data, and tau values. All the data is encoded in little-endian and needs to be handled as such by any client devices. The next step with the flex PCB is to transmit data that looks like this, which would indicate the successful implementation of BLE on the flex PCB.

7.3 Discussion

BLE marks a significant milestone in leveraging wireless communication to enhance the wearability and practicality of the transcutaneous oxygen monitor. However, to achieve this, the further steps need to be taken:

1. Solve wireless stack issue by figuring out the source of the bug that prevents CPU2 from running and sets sysevt_ready_rsp to FUS_FW_RUNNING, as shown in Figure 25.
2. Verify successful advertisement of BLE server from flex PCB to other devices, such as the data shown in Figure 27.
3. Add relevant characteristics for TOM to the GATT profile, such as number of samples per decay curve, battery operation, LED width, LED offset, TIA gain, or other settings that exist in Wired Firmware GitHub Repository.
4. Integrate the BLE server into existing TOM firmware.
5. Incorporate power-saving functionality using low power modes, such as SLEEP mode (without LSE clock you cannot go into lower power modes such as STOP/STOP2).
6. Add BLE security features as desired.

Note that the power saving and security features in steps 5 and 6 can be developed in parallel with the integration of the BLE server into the TOM firmware in step 4.

The guides and resources that have been compiled are meant to create a foundation for future development of BLE. By utilizing previous experiences, hopefully future work can overcome the remaining challenges towards making TOM wireless.

8 Graphical User Interface

A GUI significantly enhances our interaction with devices by providing an intuitive and user-friendly platform. This interface simplifies the communication between the user and the machine, enabling straightforward command inputs, real-time data reception, and clear visualization of system statuses. Specifically, within the TOM PCB project, the implementation of a GUI streamlines operations, simplifying the configuration of settings and data collection. This is a considerable improvement over command-line interfaces (CLIs), which demand that users memorize and input complex commands and syntax.

GUIs present all available options and features in an organized and visually appealing format, dramatically reducing the learning curve and improving overall usability. The simplicity of GUIs is vital in enhancing user experience, making advanced technology accessible even to those without technical expertise. Moreover, GUIs incorporate important visual cues, such as status indicators, which are essential for monitoring device activities and pinpointing issues promptly.

The integration of a GUI into device communications represents a significant advancement toward greater efficiency, user satisfaction, and engagement. Additionally, GUIs facilitate the adoption of cutting-edge technologies like Bluetooth Low Energy (BLE), boosting data collection and transmission capabilities as we move towards a more interconnected future.

8.1 GUI Implementation

To construct the GUI, the Python environment was initialized with libraries required for serial communication; PySerial is a library that allows the Python environment to access the serial ports and provides the required backend for running on various operating systems (Windows, Mac OS, Linux, and others). This serial data can be read and outputted into the terminal. To store data in Cloud storage, a Google API was implemented to allow the automated logging of tests, using OAuth2.0 as the method of accessing the servers. This section details the various sensors, the method of storing data, and a guide for setting up a test.

8.2 Methods

The integration of the GUI into the TOM device first started with understanding the needs and criteria that needed to be addressed. This required a delve into the TOM PCB and an understanding of the

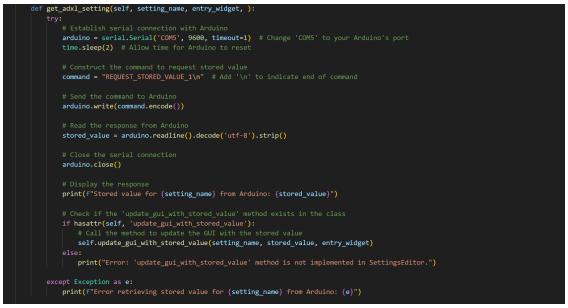
function of each of the parts at play. It also required gaining knowledge about the hardware components and software functionalities and learning about the basics of Cloud-Enabled Data storage.

8.2.1 TOM PCB and Settings

The TOM board is based on an STM32 SoC, discussed in Section 7.1; therefore, to establish a connection to it via serial communication, one needs to find the associated COM port and the transmission baud rate (BR). The COM port refers to the communication port used for serial data transmission. This is the computer port that the device connects to. Using Python's built-in serial library, the connection can be established to any peripheral connected to it. The baud rate, on the other hand, is a term used to quantify the rate of data transmission through a communication channel. Specifically for serial communication, this number indicates the maximum number of transferable bits per second. This baud rate has to be equal on both the peripheral and the computer (receiver), allowing it to sample at the proper rate. Once that is established, commands can be sent to the main MCU, which distributes the commands to their respective destination. This is done by having API calls to the specific blocks and having functions that prepare the data packet (refer to Figure 28), including the input in the correct format so that the MCU can understand the instructions. There were three main blocks on the TOM board (refer to Figure 10a) that serial setting manipulation was planned to be achieved for:

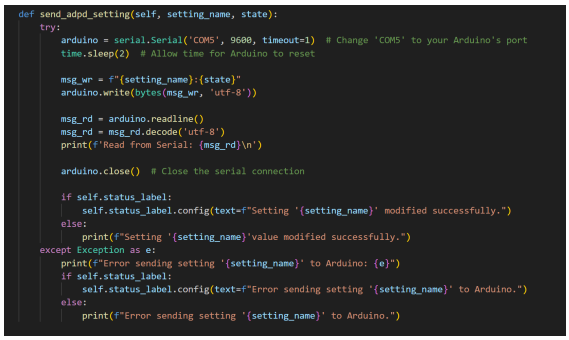
- **Accelerometer**: The accelerometer being used is helpful in correlating motion to the deviations seen in the decay time and eliminating motion artifacts.
- **Temperature Sensor**: The temperature sensor can be used to monitor the atmospheric temperature and adjust its functions accordingly.
- **Analog Front End/ADPD**: The analog front end, in general, is responsible for handling analog signals. In this case, the APDP is the photodiode used in this luminescence-based method.

These blocks and their respective settings were extracted from the board's firmware supplied by the ICAS Lab. Additionally, error checking was implemented to ensure that the inputs being sent were valid. To gain this, the list of possible values for each of the settings was obtained from the firmware, and an error-checking function was implemented to ensure that the input was valid. To maximize efficiency, a switch-case method is deployed that recognizes which specific function is being modified and checks that the value is either valid or within the specified range.



```python
def get_adxl_setting(self, setting_name, entry_widget, ):
    try:
        # Establish serial connection with Arduino
        arduino = serial.Serial('COM5', 9600, timeout=1)  # Change 'COM5' to your Arduino's port
        time.sleep(2)  # Allow time for Arduino to reset

        # Construct the command to request stored value
        command = "REQUEST_STORED_VALUE_1\n"  # Add '\n' to indicate end of command

        # Send the command to Arduino
        arduino.write(command.encode())

        # Read the response from Arduino
        stored_value = arduino.readline().decode('utf-8').strip()

        # Close the serial connection
        arduino.close()

        # Display the response
        print(f"Stored value for {setting_name} from Arduino: {stored_value}")

        # Check if the 'update_gui_with_stored_value' method exists in the class
        if hasattr(self, 'update_gui_with_stored_value'):
            # Call the method to update the GUI with the stored value
            self.update_gui_with_stored_value(setting_name, stored_value, entry_widget)
        else:
            print("Error: 'update_gui_with_stored_value' method is not implemented in SettingsEditor.")

    except Exception as e:
        print(f"Error retrieving stored value for {setting_name} from Arduino: {e}")
```

(a) API Call Code Block, showcasing the method of getting data from the TOM Board.



```python
def send_adpd_setting(self, setting_name, state):
    try:
        arduino = serial.Serial('COM5', 9600, timeout=1)  # Change 'COM5' to your Arduino's port
        time.sleep(2)  # Allow time for Arduino to reset

        msg_wr = f"{setting_name}:{state}"
        arduino.write(bytes(msg_wr, 'utf-8'))

        msg_rd = arduino.readline()
        msg_rd = msg_rd.decode('utf-8')
        print(f'Read from Serial: {msg_rd}\n')

        arduino.close()  # Close the serial connection

        if self.status_label:
            self.status_label.config(text=f"Setting '{setting_name}' modified successfully.")
        else:
            print(f"Setting '{setting_name}'value modified successfully.")
    except Exception as e:
        print(f"Error sending setting '{setting_name}' to Arduino: {e}")
        if self.status_label:
            self.status_label.config(text=f"Error sending setting '{setting_name}' to Arduino.")
        else:
            print(f"Error sending setting '{setting_name}' to Arduino.")
```

(b) API Call Code Block, showcasing the method of sending data to the TOM Board.

Figure 28: API Call Blocks. The API calls first establish a serial connection to the device, and then it checks for the different types of inputs in an if-statement. The various input types have different syntaxes for calling its value, so an if-statement is deployed to make sure it is processed and sent using the correct arguments. Upon the transmission of the data, it gets read back to the computer to ensure that the value is consistent.

8.2.2 Cloud-Based Data Storage

To establish a connection to a cloud server, the Google Drive API and OAuth frameworks were utilized. Google APIs are provided by Google and allow communication with their services and their integration into other services. These APIs are designed to be easy to use and to provide powerful functionality to developers. In this case, they were utilized as a part of a Python script to log data specifically via the Google Drive API in Section 8.4.1.

OAuth 2.0 is the authorization framework that enables applications to gain access to user accounts on these APIs. It works by delegating user authentication to the service that hosts the user account and authorizing third-party applications to access the user account. OAuth 2.0 provides authorization flows for web and desktop applications and mobile devices as shown in Figure 29:

- **Resource Center**: Typically, the end-user owns the data the application wants to access. In this case the resource center
- **Client**: The application that wants to access the user's data. The client must be registered with the provider and have a client ID and secret (Figure 32). The client in this case is the Python script.
- **Authorization Server**: The server that presents the interface where the user approves or denies the request for an access token. Here the server requesting the authorization is the OAuth2.0 server, it is integrated within the Google API Console and gives permissions and an access token to the client.
- **Resource Server**: The server hosting the protected data. Within this implementation, this server is

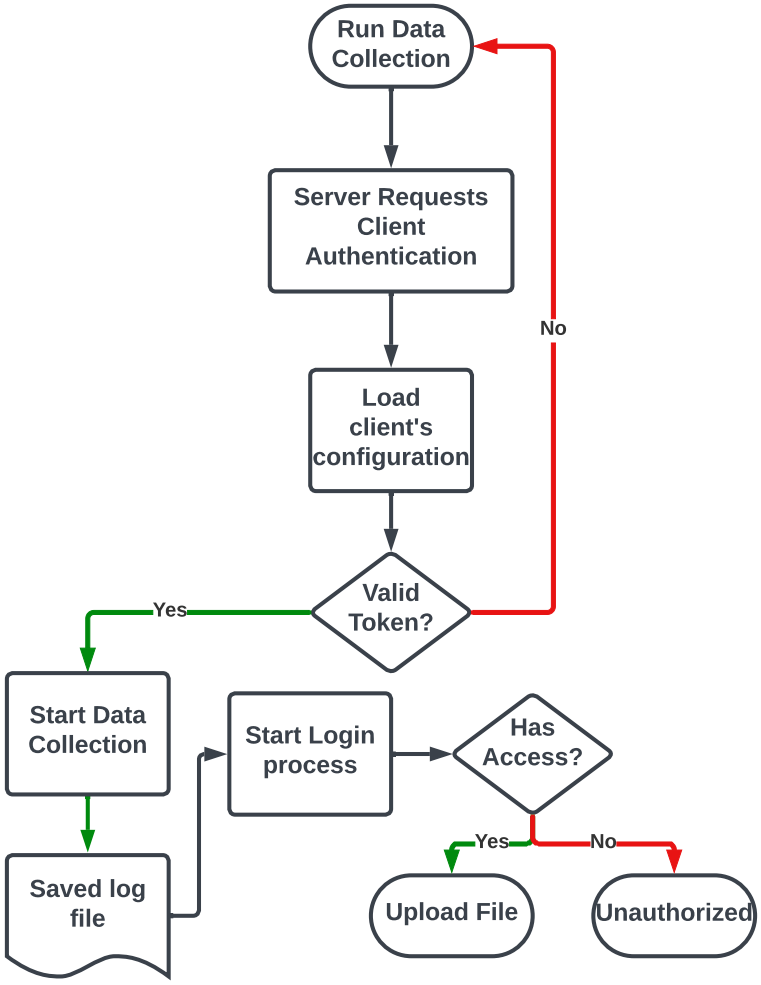


Figure 29: Data Collection and cloud storage flow. In this figure, the Google Drive server requests client authentication, which is done automatically by checking the client ID and credentials. After loading the configuration, the secret token is checked by the server, ensuring it matches the activated token (Figure 32), and the login process is initiated, where the user logs in with their Google account information.

the main Google server that redirects to the Google Drive framework

- **Access Token**: A token that is granted by the authorization server, which the client uses to access the resource server on behalf of the user. This token is stored locally and can be accessed and downloaded via the Google API Console (Figure 35).

The client will redirect the user to the authorization server, where they will log in and approve the access request. The authorization server then redirects back to the client with an authorization code, which the client will exchange for an access token. The client can then use the access token to access the resource server and request the data it needs.

8.3 Testing Procedure and Guides

This section highlights the various procedures for testing the GUI and provides examples and guides to be followed.

8.3.1 Data Collection Testing Procedure

Prerequisites:

- A code interpreter installed (Usually Visual Studio Code is used) .Alternatively the terminal can be used to run Python files.
- Gain access to the Google Drive API and be given testing credentials (This step is provisional for development purposes).
- Identify the file path and folder ID (Done by navigating to the folder in the browser and copying its ID).
- Make sure the settings are properly set up (Usually, the settings will display the current stored values).

1. Running script and initialization

This can be done by opening the script in a coding environment or running it via the terminal. Upon running, the GUI (Figure 31) should appear on the screen. Navigate to the settings menu (Figure 33) to ensure the settings are loaded and are within the desired ranges.

2. Generate Google Drive API Credentials and Key

Navigate to Google API Console 32, and create a new secret by clicking the button. A new entry shows up on the screen with a new set of secret tokens. To enable these tokens, one must download the JSON file associated that holds the token by pressing the arrow icon next to the token. Once this new file is downloaded, it should be moved to the file path that houses it within the repository (Section 8.3.1).

3. Running Data Collection

Upon pressing the data collection screen, the user will be redirected to the Google login page to authorize their client (Figure 29); this will be done by the server checking the user token and then requiring them to log in to their Google account (Section 8.3.1).

Figure 30: Google Drive Folder ID example

8.4 Results

By utilizing the aforementioned methods, a GUI was deployed for the transcutaneous oxygen monitor. This GUI has served the purpose of running data collection tests and modifying the settings of different hardware blocks on the PCB. This also allowed testing for cloud-based storage via the Google Drive API, which paves the way alongside BLE for an intuitive and all-encompassing solution. With the effectiveness of the GUI and BLE efforts, this can drive a force of contemporary capabilities and integration. The results are compiled in a GitHub repository under the ICASLab's main branch and supervision.

8.4.1 Data Collection and Logging

The collection of data was achieved through the Python environment by deploying multiple libraries that aided in establishing a serial connection with the PCB:

- **Serial**: Allows connection and manipulation of serial ports occupied on the computer.
- **OS**: Stands for Operating System and allows the user, to read, write, and manipulate system files.
- **JSON**: The JavaScript Object Notation (JSON) library allows to read and manipulate JSON files. For the GUI and GDrive authentication, the client credentials are stored as a JSON file and must be read using this library.
- **TKinter**: TKinter is the standard graphical library used to generate the GUI elements.
- **Threading**: Threading is used to allow the Python script to run 0multiple tasks at the same time by creating a separate flow of commands (thread).
- **Google API Python Client**: The main API Client given by Google. Used to establish communication between the client and a specified Google service.
- **PyDrive2**: PyDrive2.0 is a wrapper for the Google API Python Client, and it improves the efficiency of the OAuth2.0 framework.

These libraries assisted in the communication and transfer of data with the TOM board and in acquiring the correct variables for data collection and logging. Upon completion of a run, the data is automatically stored in a timestamped log file both locally and on the cloud.

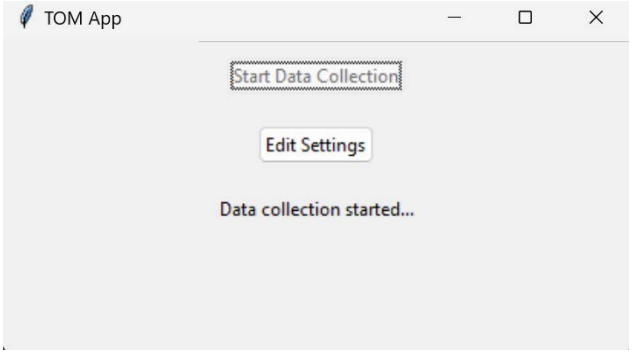


Figure 31: GUI data collection screen.

8.4.2 Cloud-Based Data Storage

Cloud computing has continued to be the path forward in this era for a multitude of different applications, and its use here for data logging has been crucial. In this project, the Google Drive API was utilized to log data to a shared drive. This was done by implementing the OAuth 2.0 framework outlined in Section 8.2.2.

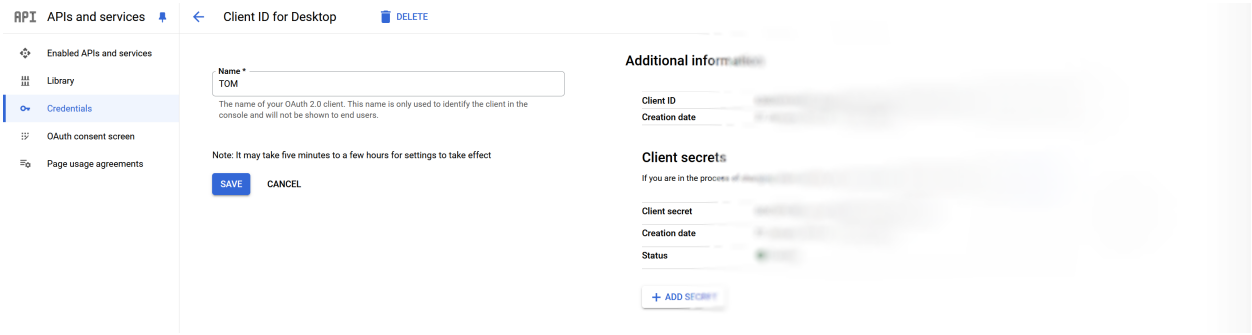


Figure 32: OAuth 2.0 screen on the Google API console shows the Client ID and secret token used in integrating the API into the software.

8.4.3 Settings Editing

One of the main functions of the device was that it was used to modify various system components via a user interface rather than writing CLI commands. This was constructed using TKinter due to its plethora of customization; multiple components in the GUI can be programmed to execute different commands. This takes user input, and then, via the functions shown in Figure 28, it sends a command serially to the transcutaneous oxygen monitor that instructs it on the exact change to be done. This is done via API calls implemented within the board's firmware, which takes the user input and addresses the specified system block.

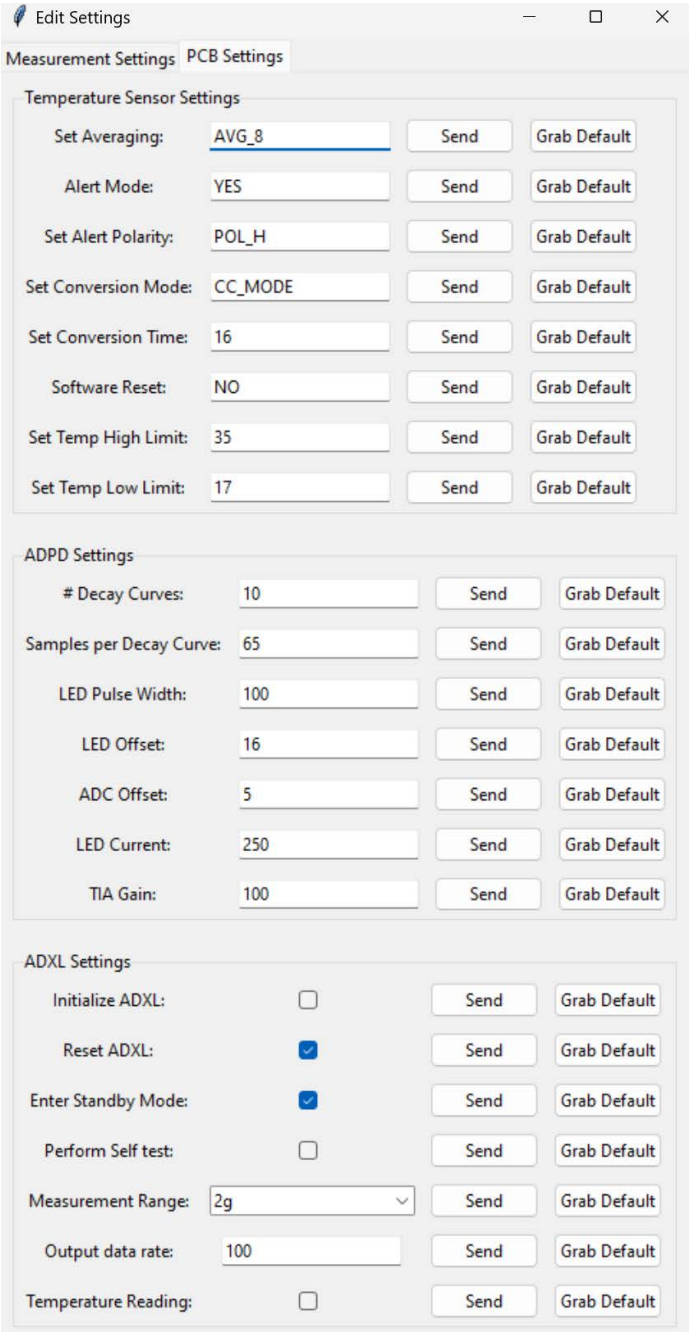


Figure 33: The transcutaneous oxygen monitor's PCB settings are filled out with the default values.

8.5 Discussion

Overall, the apparent success with achieving serial communication via a GUI (Figure 34) is a promising prospect in streamlining the efficiency of collecting data and remote patient monitoring in general. Notably, enabling the modification of various parts of the transcutaneous oxygen monitor via an appealing GUI rather than through the command line helps flatten the learning curve that comes with such low-level

system modification and assists in making this technology more widespread.

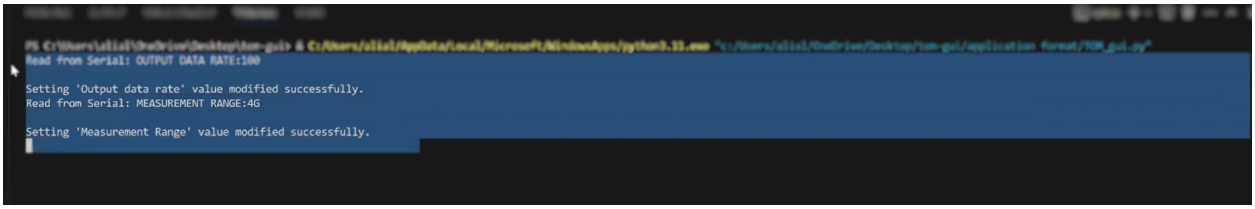


Figure 34: Serial Output Results. The monitor prints out the value being read from the serial port and notifies if the modification was successful.

The development of the GUI did come with its fair share of hardships, which ended up in aiding the understanding of not only the STM32 platform but also applications in Python, GUI development and its elements, and an understanding of serial communication and command line interactions. Most importantly, the ability to test out and use the transcutaneous oxygen monitor in itself was a challenge and pleasure as it allowed the opportunity to learn more about its inner workings, and the multiple blocks that were manipulated using the GUI.

Development challenges and limitations were faced throughout the timeline of the project. These issues hindered the development process throughout the semesters and required extra debugging to understand the underlying issues and formulate a solution:

- **Cloud Communication Issues**: In the current software, there are some issues with the connection using the current key and token. Changing the path of the repository sometimes caused the Google Authentication server to not recognize the key and broke the authentication flow. To fix this, the key and token have to be re-instantiated if the repository path changes. However, after pushing the service being used for the GUI to the public this should no longer be an issue.

Figure 35: Google Drive API Token and credentials key. These values are stored locally, and therefore, sometimes, when changing its path, the authenticator has a hard time verifying the new one.

- **Grabbing Serial Data**: Issues arose during the conception of the functions to grab values stored serially. During testing via Arduino, variables were initialized to store the values being sent serially. However, when it came to reading these values, they were not being displayed on the terminal.

- **Error Checking Statement silencing**: Due to some syntactical issues and structuring, the GUI flow sometimes bypasses the error-checking statements that are in place to validate the input. This

issue is mitigated by having corresponding error checks on the TOM board's firmware

The next logical continuation would be to work on integrating the outlined BLE framework within the GUI. This will provide a unified data collection platform that allows for both wired and wireless data transmission. Unlocking this avenue can spell wonders in the realm of transcutaneous oxygen monitoring and aid in evaluating and testing its effectiveness and viability in testing.

In conclusion, the GUI development for the transcutaneous oxygen monitor aids in improving the overall experience and the usability of the monitor. This GUI also serves as a platform for testing and improving communication, implementing frameworks such as Bluetooth Low Energy.

9 Project Challenges

In the development of advanced sensor technologies, integrating sophisticated components and ensuring their seamless interaction presents a series of challenges. Our project encompassed four different aspects: developing a motion artifact correction algorithm, quantifying the effects of motion artifact, BLE development, and GUI development. Each component not only plays a vital role in the functionality of our sensor system but also brings unique challenges that must be addressed to achieve optimal performance and reliability. The motion algorithm is the core of our sensor's capability, designed to accurately interpret and respond to physical movements. Perfecting this algorithm required overcoming significant computational and real-world application challenges. In parallel, the motion artifact testing was essential to ensure that the sensor's readings remained accurate under various dynamic conditions.

Additionally, the integration of BLE technology was critical for wireless communication, which posed challenges in ensuring robust and consistent connectivity, essential for real-time data transmission. The GUI, designed to enhance user interaction and usability, also presented its own set of challenges, particularly in aligning technical functionality with user experience. This section discusses the specific challenges we encountered in each of these areas, detailing the iterative processes and innovative solutions we employed to navigate these complexities, thereby enhancing the overall effectiveness and usability of our sensor system.

9.1 Developing Motion Artifact Correction Algorithm

The development of a motion artifact correction algorithm for a wireless transcutaneous oxygen monitor has posed several significant challenges. These challenges stemmed primarily from the complexity of distinguishing between noise induced by motion and the true physiological signals. A major challenge we faced was the variability in signal responses when testing on human subjects and under different motion scenarios. We observed that lifetime measurements varied significantly from one individual to another, which are elaborated in the literature [47], including differences in individuals' skin properties, age, and weight. This variability necessitated the implementation of more controlled testing to thoroughly understand the effects of motion artifacts on the sensor's readings.

Due to the limited data available, establishing a definitive algorithm to effectively correct for motion artifacts remains a work in progress. While the Wiener filter has provided a solid foundation, it is clear that a hybrid approach may be necessary. Such an approach would allow the algorithm to dynamically adapt to varying noise levels associated with different types of motion, enhancing the monitor's adaptability and

accuracy across a broader range of real-world applications.

9.2 Quantifying the Effects of Motion Artifact

Throughout the development of our motion artifact testing procedures for a new sensor system, our team faced several significant challenges. These challenges necessitated multiple iterations and methodological adaptations to ensure the effectiveness and reliability of our testing protocols.

One of our primary challenges was designing a robust test setup compatible with all required tests, especially as we tested the sensor independently of its BLE and GUI components. We frequently encountered issues where our setups would disconnect during specific motions, notably during high mobility tests, which are critical for assessing the sensor's performance in dynamic conditions. To address these issues, our team systematically revised our testing apparatus. Initial setups used standard connectivity solutions, which proved inadequate. Subsequent iterations involved employing more secure fixings using the gas chamber.

Based on some of the challenges noted, we also worked on developing integrated BLE and GUI components for the sensor. This integration aims to streamline future testing processes, allowing for easier setup, more stable data transmission, and enhanced user interaction during tests. By addressing the connectivity and usability challenges directly, these developments are expected to facilitate smoother testing experiences and more reliable outcomes.

9.3 BLE Development

The development journey of incorporating BLE into the transcutaneous oxygen monitor was marked by a series of technical challenges. These hurdles not only shaped the trajectory of our project but also provided invaluable learning experiences that enriched our understanding and application of wireless technology in medical devices.

9.3.1 Navigating the Complexities of BLE Technology

Gaining a comprehensive understanding of BLE posed a significant challenge due to the technology's extensive features and operational nuances. BLE's vast ecosystem, characterized by a myriad of profiles, services, and protocols, demanded an in-depth learning curve. The initial phase of our project was heavily invested in demystifying these complexities, involving rigorous study to harness BLE's full potential for our oxygen monitor.

9.3.2 Mastering the STM32 Development Environment

Similarly, the STM32 microcontroller, with its rich set of development tools and software libraries, presented a steep learning curve. The complexity of the STM32 ecosystem, while offering powerful capabilities, required dedicated effort to master. Considerable time was invested in navigating the intricacies of STM32's hardware and software interfaces, leveraging online resources, and engaging with the developer community to overcome challenges.

9.3.3 Transitioning to Integrating BLE with Custom Sensor Hardware

A significant hurdle in the project was the migration from the STM32WB55 development board to the STM32WB35 microcontroller, coupled with the integration of BLE technology into the custom hardware of the oxygen sensor. This dual-faceted challenge was marked by compatibility issues arising from the differences in the two MCU models' specifications, which necessitated adaptation of our development strategies to accommodate the STM32WB35's different features. Furthermore, embedding BLE functionality into the custom-designed sensor hardware presented its own set of obstacles. These ranged from small issues like remapping GPIO to issues that were more difficult to analyze, such as the LSE clock on the board not working. Overcoming these hurdles is still a work-in-progress and demands a flexible, iterative approach, involving extensive testing to seamlessly merge BLE capabilities with the intricacies of the custom sensor design.

9.4 GUI Development

The development of the graphical user interface presented its own unique challenges and learning curves that helped shape the path of the project and the overall integration of the GUI into the transcutaneous oxygen monitor.

9.4.1 Understanding Serial Communication

Getting a solid grasp on this part of the project meant learning the inner workings of how serial communications are established and are used to send and receive commands from whichever target it is connected to. The serial communication portion had some hurdles to its testing and integration. Firstly, testing was initiated with an Arduino board as the communication protocol is similar and translates nicely to the STM32 platform. However, immediate issues were faced with storing sent communications to the

Arduino and having it read back to the GUI. This led to spending more time understanding the different memory and storage architectures deployed within the Arduino microcontroller as well as the STM32 board to ensure no issues would occur when translating between the two. To combat this issue local variables were initialized that stored the value and were able to be sent back to the GUI upon request.

9.4.1.1 Learning Cloud-Storage Methods

When it came to managing and storing the data on a cloud system, major technical hurdles had to be overcome just by learning about the environment and setting up the client and server. To parse through this a multitude of Google's documentations were read that outlined the integration and usage of the service. This was then tested by creating a testbench environment and integrating it within a Python script that connected to a Google Drive and uploaded a test folder. After this was successfully achieved the API was then implemented within the main Python environment of the GUI

9.4.2 GUI Design

Some issues were faced when it came to designing the GUI and making sure it ran efficiently. Some issues arose from the graphical library being used; TKinter. TKinter was used to draw the interface, buttons, pop-ups, and other interface items. Some of the issues faced with this included:

- **Repetitive Code Blocks**: The GUI elements had some repetitive sections that made the code very lengthy and hard to debug. This required refactoring the code as functions to draw the GUI elements rather than repeating statements.
- **Encoding GUI Elements**: Encoding the buttons with their functions was somewhat tricky to nail down as there were multiple input methods and some of them did not support the same value-grabbing structure. For example, to grab the value of a checkbox it would be by checking its state whereas a text input would require calling a method to read the input value.
- **Updating GUI Elements**: Some issues were faced with being able to update the GUI with values being read from the serial port and distinguishing between inputs and read values.

10 Conclusion

This project vastly developed a transcutaneous oxygen monitoring system integrating advanced technologies to enhance wearable health monitoring capabilities. Throughout, we overcame various challenges such as motion artifact correction, BLE integration, and the development of a GUI. These components not only improved the functionality and reliability of the sensor but also demonstrated the potential for real-time, non-invasive monitoring of oxygen levels.

Motion artifact testing has demonstrated the reliability of the transcutaneous oxygen monitor during everyday life, which is rich with constant motion. Additionally, the integration of BLE has set the stage for future advancements in wireless and remote monitoring, offering significant implications for healthcare applications outside traditional clinical settings. Furthermore, the GUI now provides a solid foundation for more efficient and effective research and development for blood gas sensing prototypes.

This project exemplifies how engineering solutions can significantly advance medical technology, offering insights into both the challenges and potentials of integrating electronics and biomedical engineering to enhance patient care. Future work will continue to build on this foundation, aiming to introduce more sophisticated algorithms and integration with other medical monitoring devices to provide a more comprehensive suite of monitoring tools.

11 Ethical Considerations

Following ethical codes, multiple considerations must be taken into account. These considerations help not only the end user and consumers of the product but also researchers and developers create a safe environment around the device.

- **User Consent**: Users should be fully informed about the data collected by the sensor and how it will be used. Developers must obtain explicit consent from users before collecting any data, and users should have the ability to opt-out or delete their data at any time.
- **Accessibility**: Wearable sensors should be accessible to individuals with disabilities and diverse cultural backgrounds. This is also relevant for this sensor as most photometric approaches have some biases.
- **FDA Regulations**: The device must pass all FDA considerations from its accuracy in reporting results to its size and compliance with other factors.

12 Future Work

Looking forward, the project aims to expand its impact by exploring further enhancements in sensor accuracy through motion artifact correction, advancing wireless capabilities by incorporating power management schemes and security features, and adding BLE compatibility to the GUI to create a fully comprehensive user-driven development experience. These improvements will ensure that the transcutaneous oxygen monitor can meet the demands of a fast-paced research environment, and the strict constraints for wearable health technology, and provide reliable support in healthcare monitoring systems.

References

[1] World Health Organization, "The top 10 causes of death," 2019.

[2] U. Guler, I. Costanzo, and D. Sen, "Emerging blood gas monitors, how they can help with COVID-19," *IEEE Solid-State Circuits Magazine*, vol. 12, no. 4, pp. 33–47, 2020.

[3] M. Folke, L. Cernerud, M. Ekström, and B. Hök, "Critical review of non-invasive respiratory monitoring in medical care," *Medical and Biological Engineering and Computing*, vol. 41, pp. 377–383, July 2003.

[4] J. W. Severinghaus, P. Astrup, and J. F. Murray, "Blood Gas Analysis and Critical Care Medicine," *American Journal of Respiratory and Critical Care Medicine*, vol. 157, pp. S114–S122, Apr. 1998. Publisher: American Thoracic Society - AJRCCM.

[5] S. Raju, P. Sanford, S. Herman, and J. Olivier, "Postural and Ambulatory Changes in Regional Flow and Skin Perfusion," *European Journal of Vascular and Endovascular Surgery*, vol. 43, pp. 567–572, May 2012.

[6] I. Costanzo, D. Sen, L. Rhein, and U. Guler, "Respiratory monitoring: Current state of the art and future roads," *IEEE Reviews in Biomedical Engineering*, vol. 15, pp. 103–121, 2022.

[7] V. Vakhter *et al.*, "A prototype wearable device for noninvasive monitoring of transcutaneous oxygen," *IEEE Transactions on Biomedical Circuits and Systems*, 2023.

[8] D. Castaneda, A. Esparza, M. Ghamari, C. Soltanpur, and H. Nazeran, "A review on wearable photoplethysmography sensors and their potential future applications in health care," *International Journal of Biosensors & Bioelectronics*, vol. 4, no. 4, pp. 195–202, 2018.

[9] A. Temko, "Accurate Heart Rate Monitoring During Physical Exercises Using PPG," *IEEE Transactions on Biomedical Engineering*, vol. 64, pp. 2016–2024, Sept. 2017. Conference Name: IEEE Transactions on Biomedical Engineering.

[10] M. R. Ram, K. V. Madhav, E. H. Krishna, N. R. Komalla, and K. A. Reddy, "A Novel Approach for Motion Artifact Reduction in PPG Signals Based on AS-LMS Adaptive Filter," *IEEE Transactions on Instrumentation and Measurement*, vol. 61, pp. 1445–1457, May 2012. Conference Name: IEEE Transactions on Instrumentation and Measurement.

[11] R. J. Cooper, J. Selb, L. Gagnon, D. Phillip, H. W. Schytz, H. K. Iversen, M. Ashina, and D. A. Boas, "A Systematic Comparison of Motion Artifact Correction Techniques for Functional Near-Infrared Spectroscopy," *Frontiers in Neuroscience*, vol. 6, p. 147, Oct. 2012.

[12] R. Di Lorenzo, L. Pirazzoli, A. Blasi, C. Bulgarelli, Y. Hakuno, Y. Minagawa, and S. Brigadoi, "Recommendations for motion correction of infant fNIRS data applicable to multiple data sets and acquisition systems," *NeuroImage*, vol. 200, pp. 511–527, Oct. 2019.

[13] M. Kim, S. Lee, I. Dan, and S. Tak, "A deep convolutional neural network for estimating hemodynamic response function with reduction of motion artifacts in fNIRS," *Journal of Neural Engineering*, vol. 19, p. 016017, Feb. 2022. Publisher: IOP Publishing.

[14] D. R. Seshadri, E. V. Davies, E. R. Harlow, J. J. Hsu, S. C. Knighton, T. A. Walker, J. E. Voos, and C. K. Drummond, "Wearable Sensors for COVID-19: A Call to Action to Harness Our Digital Infrastructure for Remote Patient Monitoring and Virtual Assessments," *Frontiers in Digital Health*, vol. 2, 2020.

[15] I. Costanzo, D. Sen, J. Adegite, P. M. Rao, and U. Guler, "A noninvasive miniaturized transcutaneous oxygen monitor," *IEEE Transactions on Biomedical Circuits and Systems*, vol. 15, pp. 474–485, Jun. 2021.

[16] V. Vakhter, B. Kahraman, G. Bu, F. Foroozan, B. A. Beidleman, and U. Guler, "The Impact of Motion Artifacts on Transcutaneous Oxygen Measurements," in *2023 IEEE Biomedical Circuits and Systems Conference (BioCAS)*, pp. 1–5, Oct. 2023. ISSN: 2766-4465.

[17] B. Kahraman, V. Vakhter, I. Costanzo, G. Bu, F. Foroozan, and U. Guler, "A Miniaturized Prototype for Continuous Noninvasive Transcutaneous Oxygen Monitoring," in *2022 IEEE Biomedical Circuits and Systems Conference (BioCAS)*, pp. 486–490, Oct. 2022. ISSN: 2163-4025.

[18] F. Gul, O. F. Boran, R. Arslantas, F. Gul, O. F. Boran, and R. Arslantas, "Microcirculation and Hyperbaric Oxygen Treatment," in *Hyperbaric Oxygen Treatment in Research and Clinical Practice - Mechanisms of Action in Focus*, IntechOpen, Mar. 2018.

[19] R. N. Pittman, "The Circulatory System and Oxygen Transport," in *Regulation of Tissue Oxygenation*, Morgan & Claypool Life Sciences, 2011.

[20] M. M. Kmiec, H. Hou, M. Lakshmi Kuppusamy, T. M. Drews, A. M. Prabhat, S. V. Petryakov, E. Demidenko, P. E. Schaner, J. C. Buckey, A. Blank, and P. Kuppusamy, "Transcutaneous oxygen measurement in humans using a paramagnetic skin adhesive film," *Magnetic resonance in medicine*, vol. 81, pp. 781–794, Feb. 2019.

[21] I. D. Stephen, V. Coetzee, M. Law Smith, and D. I. Perrett, "Skin Blood Perfusion and Oxygenation Colour Affect Perceived Human Health," *PLoS ONE*, vol. 4, p. e5083, Apr. 2009.

[22] A. Madan, "Correlation between the levels of SpO2 and PaO2," *Lung India : Official Organ of Indian Chest Society*, vol. 34, no. 3, pp. 307–308, 2017.

[23] I. R. McPhail, L. T. Cooper, D. O. Hodge, M. E. Cabanela, and T. W. Rooke, "Transcutaneous partial pressure of oxygen after surgical wounds," *Vascular Medicine*, vol. 9, no. 2, pp. 125–127, 2004.

[24] U. K. Franzeck, A. Bollinger, R. Huch, and A. Huch, "Transcutaneous oxygen tension and capillary morphologic characteristics and density in patients with chronic venous incompetence," *Circulation*, vol. 70, pp. 806–811, Nov. 1984.

[25] R. W. Samsel and P. T. Schumacker, "Oxygen delivery to tissues," *European Respiratory Journal*, vol. 4, pp. 1258–1267, Nov. 1991. Publisher: European Respiratory Society Section: Original Articles.

[26] B. J. McGuire and T. W. Secomb, "Estimation of capillary density in human skeletal muscle based on maximal oxygen consumption rates," *American Journal of Physiology-Heart and Circulatory Physiology*, vol. 285, pp. H2382–H2391, Dec. 2003. Publisher: American Physiological Society.

[27] R. D. Restrepo, K. R. Hirst, L. Wittnebel, and R. Wettstein, "AARC Clinical Practice Guideline: Transcutaneous Monitoring of Carbon Dioxide and Oxygen: 2012," *Respiratory Care*, vol. 57, pp. 1955–1962, Nov. 2012. Publisher: Respiratory Care Section: AARC Clinical Practice Guideline.

[28] S. P. Philimon, A. K. C. Huong, and X. T. I. Ngu, "TISSUE OXYGEN LEVEL IN ACUTE AND CHRONIC WOUND: A COMPARISON STUDY," *Jurnal Teknologi*, vol. 82, June 2020. Number: 4.

[29] D. Blake, "Transcutaneous oximetry: variability in normal values for the upper and lower limb," *Journal of the South Pacific Underwater Medicine Society*, vol. 48, Mar. 2018.

[30] J. Allen and L. Rabiner, "A unified approach to short-time Fourier analysis and synthesis," *Proceedings of the IEEE*, vol. 65, pp. 1558–1564, Nov. 1977. Conference Name: Proceedings of the IEEE.

[31] "Bluetooth Wearables Are Driving the Future of Data Transfer Device Growth," May 2022.

[32] "Introduction to Bluetooth Low Energy."

[33] "STM32WB - Bluetooth, Wireless Microcontrollers (MCU) - STMicroelectronics."

[34] "Bluetooth," Apr. 2024. Page Version ID: 1220444249.

[35] "Bluetooth Technology Overview."

[36] V. Vakhter, B. Kahraman, G. Bu, F. Foroozan, and U. Guler, "A Prototype Wearable Device for Noninvasive Monitoring of Transcutaneous Oxygen," *IEEE Transactions on Biomedical Circuits and Systems*, vol. 17, pp. 323–335, Apr. 2023. Conference Name: IEEE Transactions on Biomedical Circuits and Systems.

[37] Pankaj, A. Kumar, R. Komaragiri, and M. Kumar, "A Review on Computation Methods Used in Photoplethysmography Signal Analysis for Heart Rate Estimation," *Archives of Computational Methods in Engineering*, vol. 29, pp. 921–940, Mar. 2022.

[38] Digital Signal Processing and System Theory, "Exercise "Adaptive Filters", Part 1, Wiener Filter," May 2021.

[39] "ADXL367 datasheet and product info." [Online]. Available: https://www.analog.com/en/products/adxl367.html.

[40] S. M. A. Salehizadeh, D. Dao, J. Bolkhovsky, C. Cho, Y. Mendelson, and K. H. Chon, "A Novel Time-Varying Spectral Filtering Algorithm for Reconstruction of Motion Artifact Corrupted Heart Rate Signals During Intense Physical Activities Using a Wearable Photoplethysmogram Sensor," *Sensors*, vol. 16, p. 10, Jan. 2016. Number: 1 Publisher: Multidisciplinary Digital Publishing Institute.

[41] H. Chung, H. Lee, and J. Lee, "Finite State Machine Framework for Instantaneous Heart Rate Validation Using Wearable Photoplethysmography During Intensive Exercise," *IEEE Journal of Biomedical and Health Informatics*, vol. 23, pp. 1595–1606, July 2019. Conference Name: IEEE Journal of Biomedical and Health Informatics.

[42] M. T. Islam, S. Tanvir Ahmed, I. Zabir, C. Shahnaz, and S. A. Fattah, "Cascade and parallel combination (CPC) of adaptive filters for estimating heart rate during intensive physical exercise from photoplethysmographic signal," *Healthcare Technology Letters*, vol. 5, no. 1, pp. 18–24, 2018. _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1049/htl.2017.0027.

[43] B. Roy and R. Gupta, "MoDTRAP: Improved heart rate tracking and preprocessing of motion-corrupted photoplethysmographic data for personalized healthcare," *Biomedical Signal Processing and Control*, vol. 56, p. 101676, Feb. 2020.

[44] D. Jarchi and A. J. Casson, "Description of a Database Containing Wrist PPG Signals Recorded during Physical Exercise with Both Accelerometer and Gyroscope Measures of Motion," *Data*, vol. 2, p. 1, Mar. 2017. Number: 1 Publisher: Multidisciplinary Digital Publishing Institute.

[45] S. Thakur, P. C.-P. Chao, and C.-H. Tsai, "Precision Heart Rate Estimation Using a PPG Sensor Patch Equipped with New Algorithms of Pre-Quality Checking and Hankel Decomposition," *Sensors*, vol. 23, p. 6180, Jan. 2023. Number: 13 Publisher: Multidisciplinary Digital Publishing Institute.

[46] M. Boloursaz Mashhadi, E. Asadi, M. Eskandari, S. Kiani, and F. Marvasti, "Heart Rate Tracking using Wrist-Type Photoplethysmographic (PPG) Signals during Physical Exercise with Simultaneous Accelerometry," *IEEE Signal Processing Letters*, vol. 23, pp. 227–231, Feb. 2016. Conference Name: IEEE Signal Processing Letters.

[47] A. Leonardi, C. Murphy, S. Hobson, V. Rohera, V. Vakhter, B. Kahraman, G. Bu, F. Foroozan, L. Rhein, and U. Guler, "Optimizing Transcutaneous Oxygen Measurement Sites on Humans," in *2023 45th Annual International Conference of the IEEE Engineering in Medicine & Biology Society (EMBC)*, pp. 1–4, July 2023. ISSN: 2694-0604.

[48] W. van Weteringen, T. G. Goos, T. van Essen, C. Ellenberger, J. Hayoz, R. C. J. de Jonge, I. K. M. Reiss, and P. M. Schumacher, "Novel transcutaneous sensor combining optical tcPO2 and electrochemical tcPCO2 monitoring with reflectance pulse oximetry," *Med. Biol. Eng. Comput.*, vol. 58, pp. 239–247, Feb 2020.

[49] "Cutaneous Carbon Dioxide (PcCO2) and Oxygen (PcO2) Monitors - Class II Special Controls Guidance Document for Industry and FDA," Dec. 2002.

[50] STMicroelectronics, "STM32WB Series MCU Bluetooth LE™ 5.2 IEEE 802.15.4 Zigbee 3.0 & Thread," July 2021.

[51] STMicroelectronics, "BLE Security with STM32WB - 01 Introduction and theory," June 2021.

[52] "ESP32 Wi-Fi & Bluetooth SoC | Espressif Systems."

[53] "Nordic Semiconductor Infocenter."

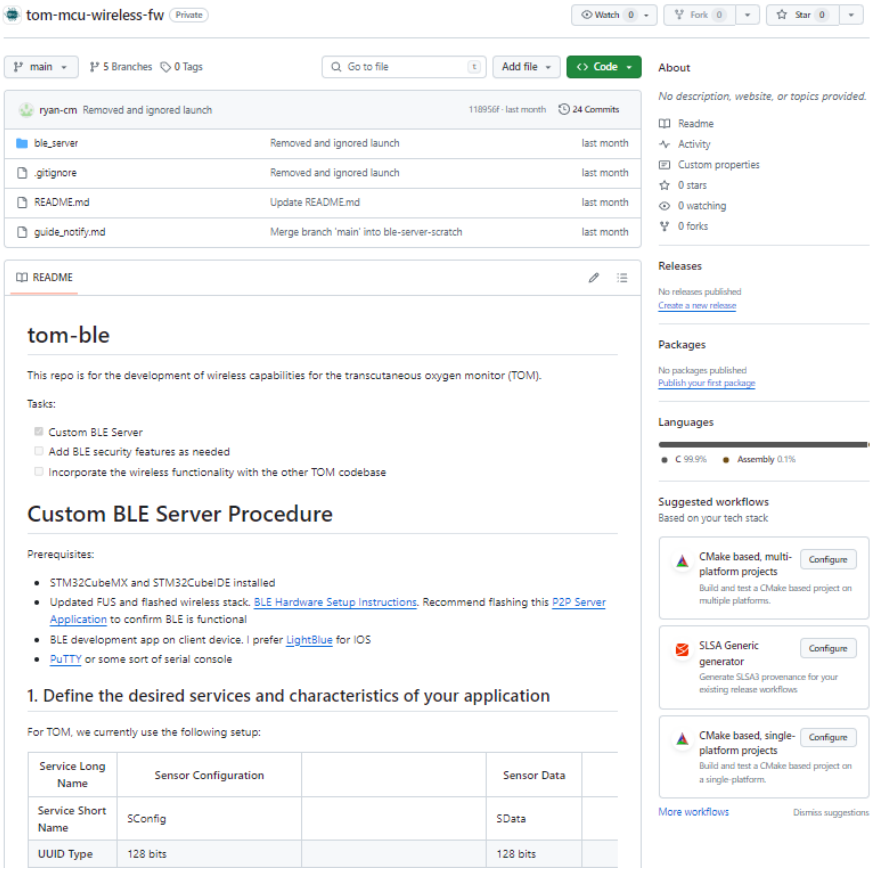
[54] "STM32 MCU MPU Software Development Tools - STMicroelectronics."

[55] "STM32CubeWB/Projects/P-NUCLEO-WB55.Nucleo/Applications at master · STMicroelectronics/STM32CubeWB."

[56] Phil's Lab, "STM32 Bluetooth Firmware Tutorial (Bring-Up) - Phil's Lab #129," Jan. 2024.

Appendices

A ICAS Lab Wireless Firmware GitHub Repository



tom-mcu-wireless-fw Private

Watch 0 Fork 0 Star 0

main 5 Branches 0 Tags Go to file t Add file Code

ryan-cm Removed and ignored launch 118956f · last month 24 Commits

ble_server	Removed and ignored launch	last month
.gitignore	Removed and ignored launch	last month
README.md	Update README.md	last month
guide_notify.md	Merge branch 'main' into ble-server-scratch	last month

README

tom-ble

This repo is for the development of wireless capabilities for the transcutaneous oxygen monitor (TOM).

Tasks:

- [x] Custom BLE Server
- [ ] Add BLE security features as needed
- [ ] Incorporate the wireless functionality with the other TOM codebase

Custom BLE Server Procedure

Prerequisites:

- STM32CubeMX and STM32CubeIDE installed
- Updated FUS and flashed wireless stack. BLE Hardware Setup Instructions. Recommend flashing this P2P Server Application to confirm BLE is functional
- BLE development app on client device. I prefer LightBlue for IOS
- PuTTY or some sort of serial console

1. Define the desired services and characteristics of your application

For TOM, we currently use the following setup:

Service Long Name	Sensor Configuration		Sensor Data	
Service Short Name	SConfig		SData	
UUID Type	128 bits		128 bits	

About

No description, website, or topics provided.

Readme
Activity
Custom properties
0 stars
0 watching
0 forks

Releases

No releases published
Create a new release

Packages

No packages published
Publish your first package

Languages

C 99.9% Assembly 0.1%

Suggested workflows
Based on your tech stack

CMake based, multi-platform projects — Configure
Build and test a CMake based project on multiple platforms.

SLSA Generic generator — Configure
Generate SLSA3 provenance for your existing release workflows

CMake based, single-platform projects — Configure
Build and test a CMake based project on a single platform.

More workflows Dismiss suggestions

B ICAS Lab GUI GitHub Repository