UCLouvain

École polytechnique de Louvain

Artifact robust real time heart rate monitoring for firefighters

Author: **Jérôme Hendrick**
Supervisor: **David Bol**
Readers: **Cristel Pelsser, Nicolas Brusselmans**
Academic year 2025–2026
Master [120] in Electrical Engineering

Acknowledgements

I would first like to thank Prof. Bol for his involvement throughout this thesis and for his valuable feedback, which helped me stay on track and continuously improve the quality of this work.

I am particularly grateful to Prof. Pelsser for introducing me to Rust in such a practical and educational way. This experience gave me a new perspective on embedded systems and played an important role in my decision to pursue embedded systems as part of my future career. The foundations I gained from her course were also invaluable for the implementation of this thesis.

My thanks also go to N. Brusselmans for his support throughout the project and for the many rich discussions and exchanges that contributed to its development.

I would also like to express my gratitude to Captain Bertrand Lorent, representing the "Zone de Secours du Brabant wallon", for his time, availability, and valuable insights into the real-world use case. Our discussions helped me better understand the practical needs and constraints of end users and guided me in adapting the system to those needs rather than expecting users to adapt to the system.

My sincere thanks go to my family and friends for their continued support and encouragement, especially during the long debugging sessions and moments of doubt. Their belief in me kept me motivated and focused throughout the research process.

I am also grateful to everyone who took the time to proofread or review my work. Their perspectives and feedback helped me improve the clarity and quality of the final manuscript.

Finally, I would like to thank all my professors and teachers for their guidance and support throughout my academic journey. Their mentorship has been invaluable in shaping my understanding of the field and preparing me for the challenges of this thesis.

Abstract

Monitoring the heart rate of firefighters during operations is a critical safety challenge, as physical effort, environmental stress, and rapid body movements can severely degrade the quality of physiological measurements. This thesis addresses this problem by developing and validating a motion-artifact-robust heart-rate tracking method as a step towards an embedded, wearable system for continuous physiological monitoring in firefighting contexts. The proposed approach combines photoplethysmographic sensing with accelerometer-based motion information and relies on a frequency-domain estimation strategy to mitigate motion-induced disturbances while preserving physiological signal content.

The work is structured in two complementary stages. First, the algorithm is implemented and evaluated in simulation using both synthetic data and the IEEEPPG benchmark dataset, which contains photoplethysmography and accelerometer recordings under dynamic motion conditions. Second, the method is ported to a real-time prototype built around an STM32f767ZI microcontroller and interfaced with a photoplethysmography sensor and an accelerometer. The prototype is then characterized in terms of estimation accuracy, task scheduling, and resource usage, enabling an assessment of its suitability for real-time operation under constrained embedded conditions.

The results show that the proposed method can provide reliable heart rate estimates in the presence of significant motion artifacts, with a mean absolute error of 2.75 BPM in steady-state conditions and 3.92 BPM during controlled physical activity. Estimation cycles complete within the specified timing constraints, while ressource usage remains well below the available microcontroller capacity. Beyond its immediate technical contribution, this thesis demonstrates the feasibility of a real-time physiological monitoring approach that could form the basis of a future wearable system for firefighter safety.

Contents

1 Background in heart rate monitoring **5**
- 1.1 Fundamentals . . . 5
  - 1.1.1 Heart rate monitoring techniques . . . 5
  - 1.1.2 Representation in the frequency domain . . . 9
- 1.2 Algorithm taxonomy . . . 10
- 1.3 Motion artifact and limitations . . . 14
- 1.4 Recommendation from previous thesis . . . 16
- 1.5 Real-world dataset . . . 17
- 1.6 State of the art . . . 18
- 1.7 This thesis . . . 24

2 Heart rate monitoring algorithm **25**
- 2.1 Design choices . . . 25
  - 2.1.1 Preprocessing and windowing . . . 26
  - 2.1.2 Reworked search range . . . 27
  - 2.1.3 Confidence estimation . . . 29
  - 2.1.4 Parameter tuning . . . 32
  - 2.1.5 Changes summary . . . 33
- 2.2 Implementation . . . 34
  - 2.2.1 Language choice . . . 34
  - 2.2.2 Estimation within a simulator . . . 36
  - 2.2.3 Estimator's features . . . 37
- 2.3 Performance metrics . . . 38
  - 2.3.1 Synthetic data performance evaluation . . . 38
  - 2.3.2 Real data performance evaluation . . . 43
- 2.4 Future improvements . . . 48
  - 2.4.1 Specific datasets for firefighters . . . 48
  - 2.4.2 Computational optimizations . . . 49
  - 2.4.3 Algorithmic improvements . . . 49
  - 2.4.4 Algorithm configuration and adaptation . . . 50
  - 2.4.5 Generator improvements . . . 50

2.5 Algorithm conclusion . . . 51

3 Prototype implementation and validation 52
3.1 Design choices . . . 52
3.1.1 Microcontroller unit platform . . . 53
3.1.2 Photoplethysmography sensor . . . 55
3.1.3 Accelerometer sensor . . . 56
3.1.4 Software language choice . . . 57
3.1.5 Embassy asynchronous runtime . . . 58
3.1.6 Modern logging with defmt . . . 58
3.1.7 Toolchain and build support . . . 59
3.2 Implementation . . . 59
3.2.1 Nucleo clock setup . . . 59
3.2.2 Prototype assembly and wiring . . . 61
3.2.3 Sensor interfacing . . . 64
3.2.4 Sensor setup . . . 65
3.2.5 Prototype peripheral usage . . . 66
3.2.6 Task design and scheduling . . . 67
3.3 Characterization . . . 69
3.3.1 Heart rate estimation performance . . . 69
3.3.2 Task execution timing . . . 73
3.3.3 Resource usage . . . 74
3.4 Future improvements . . . 75
3.4.1 Enhance stability and robustness . . . 76
3.4.2 Evolve into a wearable form factor . . . 76
3.4.3 Use hardware acceleration . . . 76
3.4.4 Grant wireless monitoring capabilities . . . 77
3.4.5 Floating point representation . . . 77
3.5 Prototype conclusion . . . 77

4 Conclusion 78

References 81

A Use of artificial intelligence 84
A.1 Overview of artificial intelligence tool usage . . . 84
A.2 Tools employed . . . 84
A.3 Specific applications of artificial intelligence (AI) tools . . . 84
A.3.1 Code development and review . . . 84
A.3.2 Writing and communication . . . 85
A.3.3 Technical problem solving . . . 85

A.4 Boundaries and limitations 85
A.5 Verification of artificial intelligence (AI)-assisted outputs 86
A.6 Transparency statement 86

List of Abbreviations

ACC ACCelerometer

ADC Analog-to-Digital Converter

AI Artificial Intelligence

BLE Bluetooth Low Energy

BPM Beats Per Minute

CNN Convolutional Neural Network

CPU Central Processing Unit

CSV Comma-Separated Values

DFT Discrete Fourier Transform

DMA Direct Memory Access

DSP Digital Signal Processing

ECG ElectroCardioGram

FF FireFighter

FIFO First In First Out

HAL Hardware Abstraction Layer

HR Heart Rate

I/O Input/Output

I²C Inter-Integrated Circuit

IC Integrated Circuit

IDE Integrated Development Environment

IoT Internet Of Things

IRQ Interrupt ReQuest

LED Light-Emitting Diode

MA Motion Artifact

MAE Mean Absolute Error

MCU MicroController Unit

PCB Printed Circuit Board

PLL Phase-Locked Loop

PPG PhotoPlethysmoGraphy

PSD Power Spectral Density

PWM Pulse-Width Modulation

RAM Random Access Memory

RMSE Root Mean Square Error

RNN Recurrent Neural Network

RTOS Real-Time Operating System

SDK Software Development Kit

SNR Signal-to-Noise Ratio

SPI Serial Peripheral Interface

SWD Serial Wire Debug

UART Universal Asynchronous Receiver-Transmitter

List of Figures

1.1 Working principle and common positions for PPG sensors. 7
1.2 Comparison of ECG and PPG signals [16]. 8
1.3 Labels of the different parts of the ECG signal [17]. 8
1.4 Link between a DFT and a spectrogram. 10
1.5 Comparison of different HR estimation algorithms on the same signal without MA. 13
1.6 Comparison of different HR estimation algorithms on the same signal with MA. 15
1.7 Example of IEEEPPG dataset recordings. 18
1.8 Temko et al. algorithm pipeline [19]. 19

2.1 Pipeline of the proposed algorithm. ■ boxes indicate unchanged steps, ■ boxes indicate improved steps, and ■ boxes indicate new steps. 26
2.2 Hann window coefficients. 27
2.3 Examples of shaping functions. 28
2.4 Deviation score examples. 30
2.5 Dissipation score examples. 31
2.6 Block diagram of the algorithm validation framework. 37
2.7 Synthetic data created by the generator. 39
2.8 Algorithm's spectra for synthetic data. 41
2.9 Spectrograms of the algorithm's input for the synthetic data. 42
2.10 Examples of accurate estimations. 44
2.11 Examples of estimation errors. 45

3.1 Nucleo-144 STM32F767ZI development board. 55
3.2 MAX30101WING breakout board for the MAX30101 PPG sensor. . 56
3.3 X-NUCLEO-IKS01A3 motion MEMS and environmental sensor expansion board. 57
3.4 Nucleo clock configuration diagram. 60
3.5 Prototype assembly. 62
3.6 MAX30101WING PPG in use. 62

3.7 Prototype PCB blocks and their exposed pins. 63
3.8 Data collection and processing tasks pipeline. 67
3.9 Simulated waterfall diagram of the embedded codebase. 68
3.10 PSS spectrum and HR estimates in a steady state. 71
3.11 PSS spectrum and HR estimates under exercise. 72
3.12 Measured waterfall diagram from task traces. 73

List of Tables

1.1 Results of Temko et al.'s algorithm on the IEEEPPG dataset [19]. . 23

2.1 Comparison of implementation language candidates. 35
2.2 Performance metrics for the synthetic data examples. 43
2.3 Mean absolute error and confidence score for each run on the IEEEPPG training and testing datasets. 47

3.1 Prototype CPU usage. 74
3.2 Prototype memory resource usage. 75

Introduction

Context

In fields where timings are critical, stakes are high and the environment is hostile, there are persistent threats to the safety of the deployed personnel. In such cases, the ability to monitor vitals while their tasks are ongoing is a valuable asset.

Among these high intensity fields, firefighting stands out as one of the most dangerous. Because one of the primary missions of firefighters (FFs) is to manage life-threatening situations, they are exposed to many hazardous conditions such as smoke inhalation, extreme heat and debris impact among others. Due to these conditions, FFs experience non-fatal injuries at rates between 1.4 and 7.1 times higher compared to other professions in the USA [1]. For these injuries, "SOII reports over half (52%) of injuries and illnesses resulting in lost workdays occur due to overexertion and body reaction. Slips, trips, and falls are second at 21%." [2].

This concern is reinforced by this summer's wildfire activity in Europe. According to EFFIS estimates, several countries already exceeded their long-term burned-area averages (period $2006 - 2025$): France reached about $95,284$ ha in 2026 versus an annual average of $14,756$ ha, and Spain about $283,988$ ha versus $95,246$ ha on average [3]. Such large-scale and prolonged interventions increase thermal strain, fatigue and operational pressure on FFs, strengthening the case for continuous, in-field physiological monitoring to detect overexertion early and support safer decision-making.

While these risks obviously threaten the health of FFs, they also reduce the efficiency of their services and overall increase costs for comparable response effectiveness. In the USA, "The estimated cost of firefighter injury is estimated to range between \$1.6 billion and \$5.9 billion annually" [2]. This reasonably motivates states and organizations to invest in solutions, beyond the obvious ethical question of protecting the health of their personnel.

In order to mitigate such risks, FFs are trained to recognize dangerous situations and to react accordingly. However, in the presence of adrenaline and stress, this training can be skewed and lead to mistakes. In order to help FFs accurately estimate their body's state, monitoring their vitals is a strong candidate to improve

their safety.

This thesis work

This thesis work is a continuation of the work done by Martin Brans in his Master's thesis at UCLouvain [4]. In his work, he discussed the challenges of monitoring vitals accurately, especially the heart rate (HR), in the presence of movement. He also presented in detail the existing FFs safety equipments and their current limitations. Finally, he began implementing an algorithm on a prototype to validate the performance of his solution. This development led to a platform able to measure HR in real time and send it via Bluetooth low energy (BLE). However, the included algorithm was not robust to movement and the estimations would diverge from the real HR in the presence of such disturbances.

This thesis presents the design and deployment of a more motion artifact (MA)-robust algorithm, along with its validation on a prototype. To take advantage of the conclusions of the previous thesis, the algorithm was first implemented in a simulator to facilitate its testing and debugging. After passing performance tests on synthetic and real-world data in the simulator, the algorithm was implemented on a prototype and validated under lab conditions using live sensor data.

Tools used

The development of this whole project was done using Visual Studio Code [5], a free and open-source integrated development environment (IDE) developed by Microsoft. It is a powerful and versatile tool thanks to its large ecosystem of extensions, making it ideal for any project requiring multiple languages and tools.

For version control, we used Git [6] and hosted the repositories on GitHub [7]. We followed the Gitflow workflow [8] to ensure a clean and organized repository structure and to keep a working state available for parity testing. This also simplified the revert process in the event of a bug or issue.

Almost all of the central code developed for this thesis is written in Rust [9]. This choice was motivated by the language's emphasis on reliability and embedded real-time readiness, with performance comparable to C/C++ [10] while still offering a higher level of abstraction and safety. These features, combined with the large ecosystem of libraries and tools available, made it a strong candidate for this project.

The remaining codebase used to perform data formatting, visualization and the computation of one constant is written in Python [11] because of its simplicity and the large library ecosystem that simplifies these tasks.

In order to ensure the reproducibility of the builds and tests regardless of the local environment, we used Docker [12] to containerize the development environment. This allows anyone to run the code and obtain the same results as on the development machine without having to manage dependency installation or environment configuration.

For the prototype, we used a STM32 Nucleo F767 development board [13]. This kit is part of the STM32 ecosystem that allows for solderless prototyping thanks to its header pins and compatible shield modules. For the development module not compatible with the pinout of these headers, a breadboard was used to connect it to the Nucleo. This Nucleo was chosen due to its powerful processor, large memory and many available peripherals. A more restricted board will likely be used in the future, but the prototype was designed with plenty of spare resources to run the algorithm, and the optimization process was left for later iterations.

Structure

This document is organized in four chapters.

For readability, this manuscript refers to the HR estimator crate as *HeartTrack* (implemented as `hearttrack-rs`) and to the embedded runner as *EmberGuard* (implemented as `emberguard-vest`).

Chapter 1 presents in more detail the research problem addressed in this thesis, presents the theoretical background necessary for understanding the proposed approach, and reviews the state of the art in HR monitoring algorithms.

Chapter 2 presents how the MA robust HR tracking algorithm works. It then describes how it was validated on synthetic and real-world data, and how it was implemented as a ready-to-use crate (library) in Rust.

Chapter 3 presents the prototype built to test the algorithm with live data and quantify its real-time constraints. It also describes the hardware selection and the choices made for the sensor interface. It concludes by presenting some of the results obtained with the prototype and how they compare to the results comming from the simulated environment.

Finally, the conclusion summarizes the contributions of this work and discusses future perspectives for extending and improving the proposed solution.

Transparency and compliance

In accordance with institutional guidelines, this thesis includes a detailed account of all artificial intelligence tools used during its preparation. Appendix A provides a comprehensive disclosure of the generative AI tools employed, their specific

applications, and the boundaries observed to maintain academic integrity. This transparency ensures compliance with UCLouvain's AI governance framework and enables readers to assess the authenticity and originality of the work.

Chapter 1

Background in heart rate monitoring

This chapter provides the background needed to understand the context, motivation, and central concepts of this thesis. It covers the fundamentals of HR monitoring, the taxonomy of algorithms used for this purpose, the challenges posed by MA, along with a guide explaining how to read common spectral analysis representations and a review of previous work in the field. Finally, it identifies the gap in the state of the art that this thesis aims to fill.

Some of the content of this chapter is based on the findings and conclusions of the previous works this thesis is built upon. More details can be found in Martin Brans' Master's thesis at UCLouvain [4].

1.1 Fundamentals

1.1.1 Heart rate monitoring techniques

The heart is a muscular pump with four chambers that contract rhythmically to circulate blood. The HR, expressed in beats per minute (BPM), and its derived statistics are widely used indicators of cardiovascular condition, physical exertion and stress. Sustained elevations or sudden spikes in HR can signal high workload or acute stress, while unexpected drops may indicate fatigue or other problems.

Building on the findings of the previous thesis [4], which identified HR as the most relevant parameter to monitor for FFs, this work therefore focuses on implementing a continuous, real-time HR monitoring as the primary metric for intervention support.

To estimate the HR, various techniques have been developed. As in all engineering disciplines, there is no perfect solution, and each technique has its own

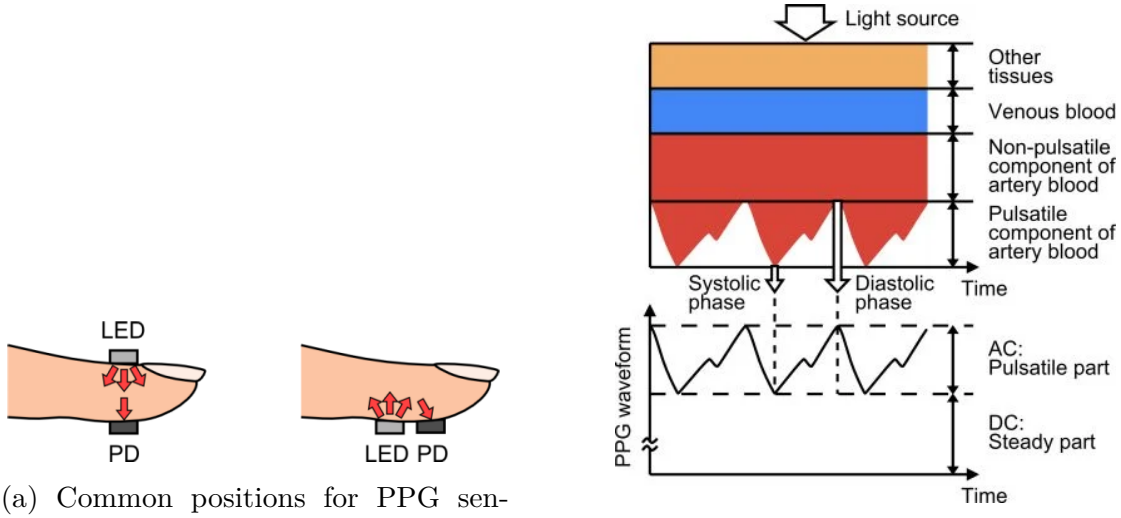
strengths and limitations.

The most accurate and precise technique widely used in the medical field is the electrocardiogram (ECG), which measures the electrical activity of the heart using electrodes placed on the skin. While it provides reliable HR measurements, it is invasive as it requires electrodes to be attached to the skin. Gel-based electrodes are used to ensure good contact and signal quality, "but it makes them uncomfortable and too slow to put on" [4]. Using electrodes without gel is possible, but "their impedance is much higher, and any slight movement relative to the skin causes a lot of *(signal)* noise" [4].

There are also less invasive esoteric techniques such as ballistocardiography, which measures the mechanical activity of the heart by detecting the body's movements caused by the heartbeat and seismocardiography, which measures the vibrations produced by the heart's activity. While these techniques are less invasive than ECG setups, they exhibit an even higher sensitivity to noise caused by movement of the sensor chain relative to the skin, which makes them unsuitable for FFs.

Due to the specific requirements of FFs, the ideal monitoring solution should be non-invasive, comfortable to wear, quick to deploy, and sufficiently unobtrusive to operate during intense motion and in adverse conditions. In this context, photoplethysmography (PPG) stands out as the most interesting option for continuous, wearable vitals monitoring. It can be integrated into compact devices without the need for gel electrodes or extensive skin preparation, and it works by emitting light into the skin under monitoring and measuring its reflectivity [14]. Since the heart pumps blood rhythmically, the density of blood in the skin changes accordingly, which also affects the perceived color of the skin and the intensity of the reflected light. The HR and other cardiovascular parameters can therefore be extracted from this raw signal by analyzing the variations in the reflected light [15]. As a bonus, signals obtained from PPG sensors can also be used to extract other parameters such as blood oxygen saturation, respiratory rate and stress levels, which makes it an even more interesting candidate for broader vitals monitoring.

Figure 1.1 shows the working principle and common positions for PPG sensors. The light-emitting diode (LED) emits light towards the skin, which is then captured by the photodetector. The intensity of this light varies with the blood volume in the skin, which is modulated by the heartbeat.



(a) Common positions for PPG sensors [15].

(b) Working principle of PPG [15].

Figure 1.1: Working principle and common positions for PPG sensors.

In the transmission mode, the photodetector is placed on the opposite side of the LED, where as in reflectance mode both are placed on the same side of the skin. Transmission is typically more accurate but also more invasive and less comfortable, making it unsuitable for FFs. This thesis thus adopts the more comfortable, if slightly less accurate, reflectance setup.

Regardless of the mode, an increased blood volume leads to greater light absorption by the skin, resulting in a decrease in the raw photodetector signal. Many sensor systems invert or normalize this signal for display, which flips the apparent polarity of the PPG waveform. Discarding the DC component of the signal due to the constant blood volume and skin absorption, the remaining AC component contains the HR signal.

Figure 1.2 shows how the PPG and ECG raw signals are correlated.

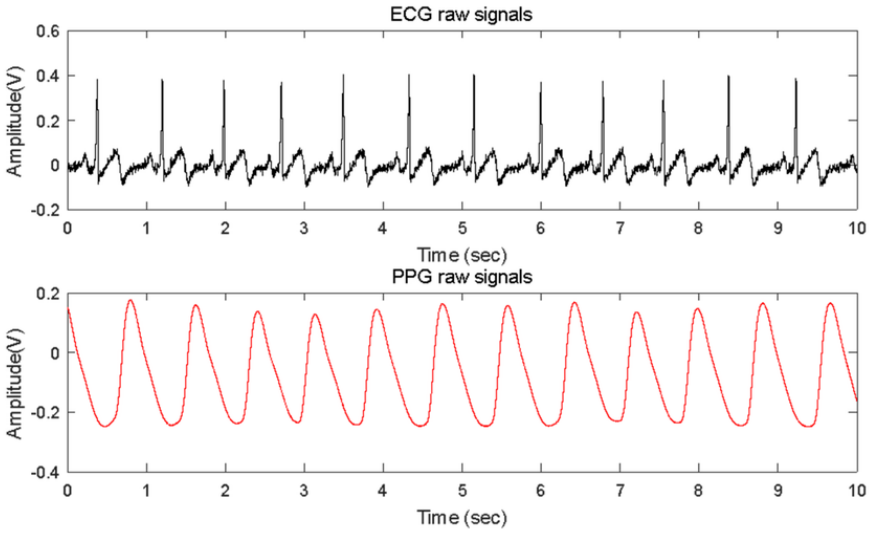


Figure 1.2: Comparison of ECG and PPG signals [16].

The human circulatory system can be modeled as a dispersive low-pass system: the higher-frequency components of the pulse are attenuated, broadening the PPG peaks and making them more sinusoidal, and the overall delay ($\approx$ 100 ms in this case) can be interpreted as the phase response of the circulatory system's transfer function rather than as a direct consequence of amplitude attenuation alone.

The PQRST taxonomy for ECG HR analysis is summarized in Figure 1.3.

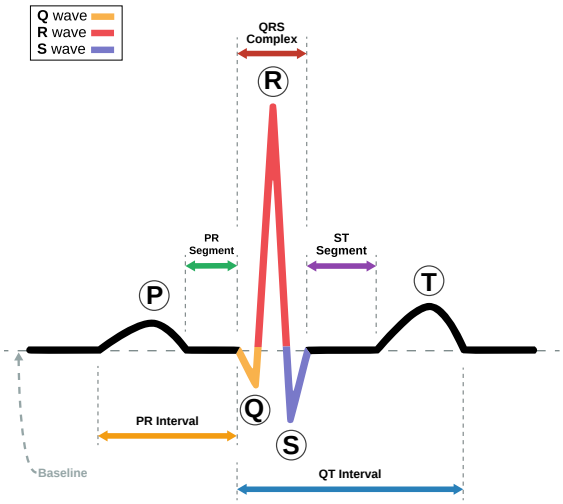


Figure 1.3: Labels of the different parts of the ECG signal [17].

The ECG R-peak marks electrical depolarization, which precedes the mechanical pulse. The PPG spike therefore lags behind it by the pulse transit time. If the PPG signal's polarity is reversed relative to the raw sensor output, increased blood volume absorbs more light and therefore appears as an upward deflection on the plotted trace. Conversely, the PPG often reaches a relative minimum around the ECG T-wave because skin blood volume is lower during this phase of the cardiac cycle.

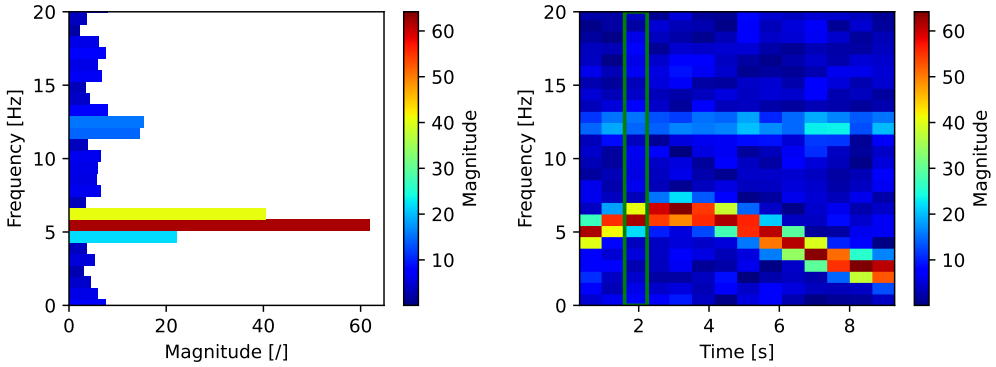
The ECG signal is considered the ground truth throughout this thesis unless otherwise specified.

1.1.2 Representation in the frequency domain

A convenient way to visualize how the spectral content of a signal evolves over time is through a spectrogram. Unlike a conventional Fourier transform, which provides the frequency content of the entire signal at once, a spectrogram preserves temporal information by repeatedly computing the frequency spectrum over short time windows. It is therefore particularly well suited for the analysis of non-stationary signals such as in PPG, whose frequency content changes due to MAs, physiological events or natural variations in the HR.

The spectrogram is obtained by dividing the signal into short, overlapping time windows and computing the Fourier transform of each window independently. The resulting spectra are then displayed as a two-dimensional representation, where the horizontal axis represents time, the vertical axis represents frequency, and the color intensity indicates the magnitude of each frequency component.

Figure 1.4 illustrates how a spectrogram is constructed from multiple discrete Fourier transform (DFT) computations. Figure 1.4a represents the DFT of the green-outlined third window of the spectrogram shown in figure 1.4b.



(a) DFT example.

(b) Spectrogram example.

Figure 1.4: Link between a DFT and a spectrogram.

This DFT plot uses swapped axes compared to the conventional DFT representation to match the y axis content of the spectrogram. This DFT axis convention is maintained throughout this thesis for better parallel with the spectrograms.

In this toy example, the dominant frequency varies over time and is near 5 Hz in the window detailed in figure 1.4a.

As with any time-frequency analysis, the spectrogram exhibits a trade-off between temporal and frequency resolution. Short analysis windows provide a more accurate localization of transient events but reduce the ability to distinguish closely spaced frequency components. Conversely, longer analysis windows improve frequency resolution at the expense of temporal precision. The choice of window length therefore depends on the characteristics of the signal under analysis and the specific phenomena of interest.

In subsequent spectrograms, the x axis will be scaled accordingly, as the analysis windows will be wider and the frequency axis may be expressed in BPM. Besides these scaling differences, the underlying methodology remains unchanged.

1.2 Algorithm taxonomy

Once the raw signal is obtained from the chosen sensor strategy, it needs to be processed to extract an estimate of the HR. Many different types of algorithms can be used for this purpose, as there is no perfect solution and each approach involves trade-offs. These algorithms can generally be classified into the following categories.

Peak detection algorithms work by identifying the highest local points in raw

signals and matching them with individual heartbeats. This category is simple and computationally efficient, but is highly sensitive to noise, which can introduce false peaks and lead to inaccurate HR estimates, even when pre-processing techniques such as band-pass filtering or outlier removal are applied.

An alternative to peak detection is threshold-based detection. This approach works in a similar manner, but instead of relying on local maxima, it identifies points where the signal crosses a predefined threshold, reducing the impact of noise on the estimation. This category is generally considered more robust to artifacts, but it still suffers from the same limitations as peak detection algorithms.

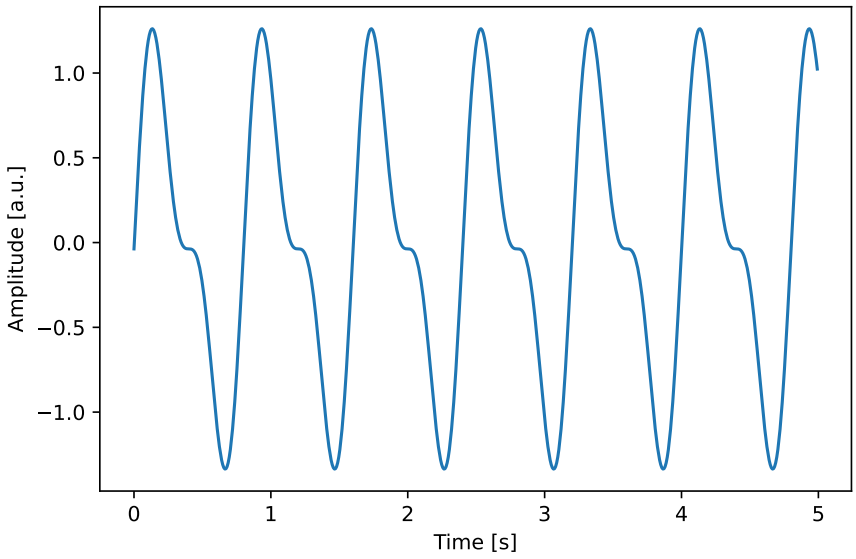
Both peak detection and threshold-based algorithms operate in the time domain: they filter the signal and then either measure the intervals between detected events or count events within a given time window to estimate the HR. They are generally computationally efficient and can be implemented on low-power microcontrollers, but they are remain highly sensitive to noise, which can cause false detections and lead to inaccurate HR estimates.

Machine learning-based algorithms use signal decomposition and pattern recognition techniques to extract features from raw signals, which are then fed into supervised models such as convolutional neural networks (CNNs) or recurrent neural networks (RNNs) to estimate HRs. These algorithms are incredibly powerful and can achieve high accuracy, but their performance strongly depends on their ability to decompose the signal into its different components, which can be challenging when the noise cannot be easily modeled. Additionally, they require large amounts of data to train and are computationally intensive, which limits their ability to run in real-time on the edge and often requires offloading preprocessed data to a more powerful central compute node.

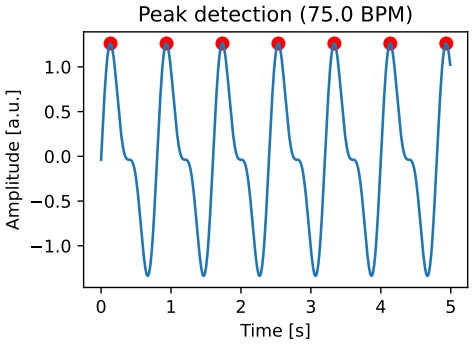
Spectrum-based algorithms take a radically different approach. Instead of attempting to identify events or patterns in the time-domain, they transform the signal into the frequency domain after pre-processing and perform the analysis in that domain. After further signal conditioning and optional compensation, the HR is estimated by identifying the frequency bin with the highest power, which should correspond to the HR frequency. These algorithms are generally more robust to noise, as noise power is typically distributed across the spectrum, while the HR power is concentrated in a single bin. However, they generally perform worse for non-stationary signals, in which case they usually require longer measurement windows, which in turn reduces the responsiveness of the estimation. Additionally, they are more computationally intensive than time-domain algorithms due to the domain transformation and filtering steps, which means they must be carefully

optimized to run in real-time on edge devices.

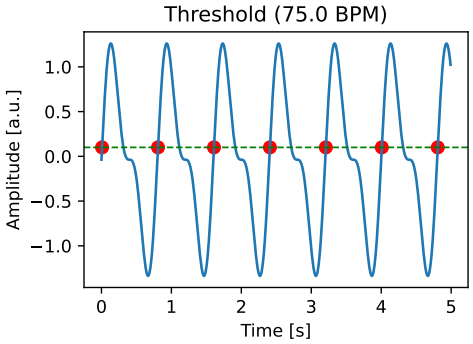
Figure 1.5 presents one internal state representation for each estimation category. Note that figure 1.5d is included only as a conceptual illustration, since no CNN-based implementation is developed in this thesis. The estimated HR is indicated in the title of each subfigure. All examples are generated from the synthetic PPG signal defined in equation (1.1), using $f_{HR} = 1.25$ Hz (75 BPM) and without additive noise.



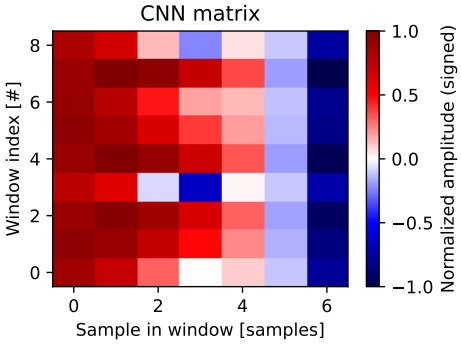
(a) Original signal without MA



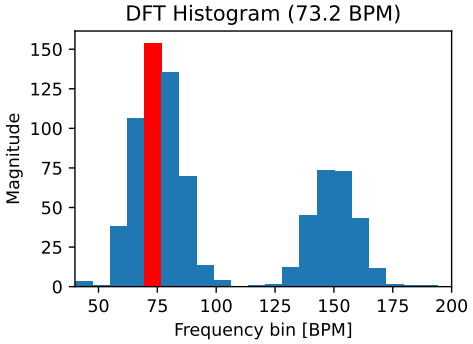
(b) Peak detection algorithm.



(c) Threshold-based algorithm.



(d) CNN-based algorithm.



(e) Spectrum-based algorithm.

Figure 1.5: Comparison of different HR estimation algorithms on the same signal without MA.

$$PPG(t) = 1 \cdot \sin(2\pi \cdot f_{HR} \cdot t) + 0.5 \cdot \sin(4\pi \cdot f_{HR} \cdot t) \tag{1.1}$$

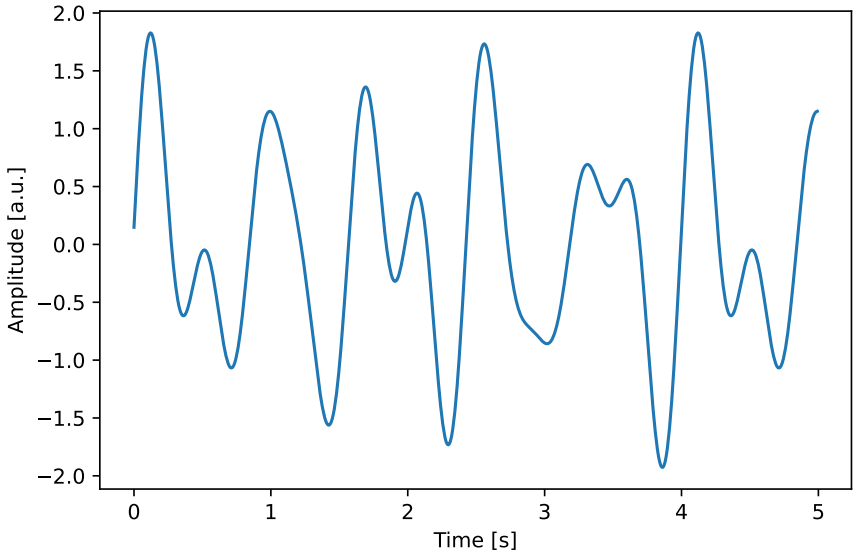
The signal model in equation (1.1) consists of a fundamental cardiac component at frequency f_{HR} and a second harmonic at $2f_{HR}$. The second harmonic is assigned half the amplitude of the fundamental to reproduce the rounded, slightly asymmetric morphology commonly observed in PPG waveforms. For clarity, the model intentionally excludes both the DC component and measurement noise.

As expected, all algorithms perform well on this ideal input signal. The slight difference observed in the estimated HR obtained with the spectrum-based algorithm is due to the fact that the conversion from the time-domain to the frequency domain is not continuous. Consequently, the frequency bins are discrete and their center frequencies do not necessarily correspond to the actual HR, which can lead to a slight HR estimation error depending on the actual signal frequency and the spectral resolution. Further details about this spectral limitation are discussed in Section 1.6.

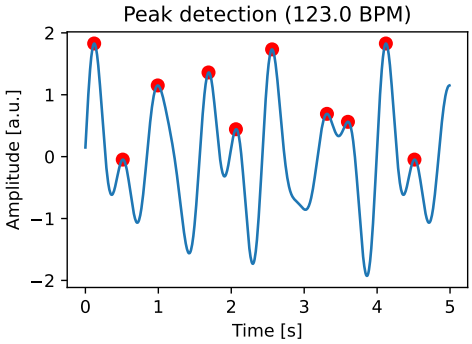
1.3 Motion artifact and limitations

As soon as the subject moves away from the ideal case of being perfectly still in a controlled environment, the performance of the algorithms degrades significantly. This is due to the additional noise introduced in the measurement chain, commonly referred to as MA. These artifacts are no longer simply random white noise. They exhibit specific patterns that are hard to model and can sometimes be confused with actual heartbeat signals, leading to erroneous estimates. This source of inaccuracy is particularly problematic for FFs, as they are often engaged in intense physical activity, making the presence of high energy MA almost unavoidable during operation.

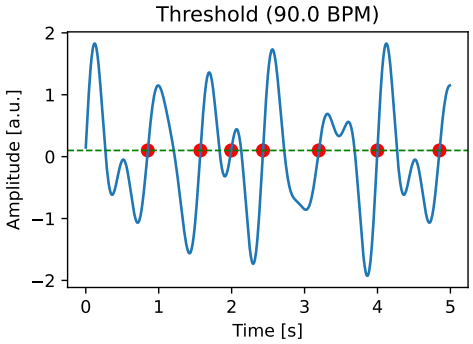
Figure 1.6 presents the same internal representations as in figure 1.5, but this time with an input signal containing strong MAs, generated from equation (1.2) with $f_{MA} = 2$ Hz (120 BPM). This value of MA frequency is chosen because it corresponds to a common arm swing frequency while running.



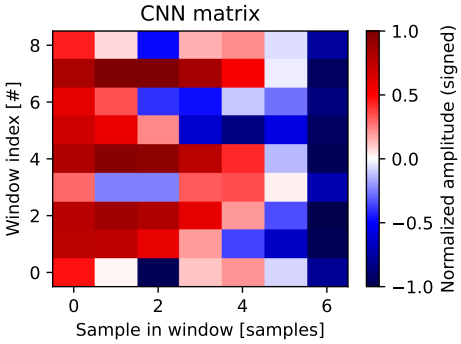
(a) Original signal with MA



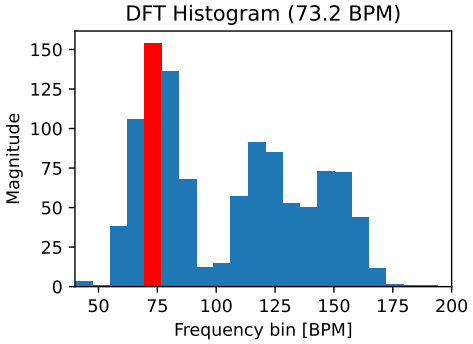
(b) Peak detection algorithm.



(c) Threshold-based algorithm.



(d) CNN-based algorithm.



(e) Spectrum-based algorithm.

Figure 1.6: Comparison of different HR estimation algorithms on the same signal with MA.

$$PPG(t) = 1 \cdot \sin(2\pi \cdot f_{HR} \cdot t) + 0.5 \cdot \sin(2\pi \cdot f_{HR} \cdot t) + 0.6 \cdot \sin(2\pi \cdot f_{MA} \cdot t) \tag{1.2}$$

As can be seen, the performance of all algorithms degrades significantly, with peak detection and threshold-based algorithms being the most affected, due to their high sensitivity to noise. This limitation does not result from an inadequate parameter heuristic. Regardless of the chosen threshold of figure 1.6c, no accurate prediction can be obtained because of interference from MAs which have a higher amplitude than the original HR signal.

The machine learning-based algorithm also suffers from the presence of MAs, as it relies on the ability to extract meaningful features from the raw signal, which can be difficult when the noise cannot be easily modelled. As a reminder, both figure 1.5d and figure 1.6d do not represent actual CNN outputs and are included for visualization purposes only.

The spectrum-based algorithm is still able to distinguish the true HR frequency bin from the MA-related bins because they are of smaller amplitudes. If the MA signal had a higher energy than the HR signal, the spectrum-based algorithm would also fail to provide an accurate estimation as it would identify the strongest spectral peak, which would be caused by MAs.

1.4 Recommendation from previous thesis

In his thesis, Martin Brans investigated wearable HR tracking techniques for firefighters [4] and concluded:

1. Requirements for usability and rapid deployment rule out the use of gel-based clinical electrodes.
2. MAs are the primary obstacle to reliable HR estimations under real-world conditions.
3. Algorithms that perform well under mild motion in the laboratory often fail under real-world conditions

He also developed and tested a prototype using a wearable PPG sensor. His prototype relies on an enhanced peak detection algorithm, which operates in the time domain, and is therefore highly sensitive to MAs. As a result, the prototype performed poorly in real-world conditions, which is consistent with the conclusions of his thesis.

He recommends focusing on sensor fusion, realistic motion datasets, and evaluation protocols that accurately reflect real-world intervention scenarios [4].

1.5 Real-world dataset

Before being tested real-time data from FFs, algorithms must first be evaluated on a dataset with ground-truth HR values. Fortunately, such a dataset already exists [18] and is well suited for this purpose. The IEEEPPG dataset was originally recorded and used for the 2015 IEEE Signal Processing Cup, whose theme was "Heart Rate Monitoring During Physical Exercise Using Wrist-Type Photoplethysmographic (PPG) Signals". Since then, this dataset has become a standard benchmark for evaluating HR estimation algorithms. It contains two PPG channels acquired using green light (515 nm) and three orthogonal accelerometer (ACC) channels. All channels are sampled at 125 Hz. For each recording, there is also a gel-based ECG HR trace considered as the ground truth.

In the latest version, the dataset contains 12 recordings in a "train" set containing sufficient MA for the algorithm development, and 6 other additional recordings in a "test" set acquired under similar conditions but with significantly more severe MAs to test the algorithm. Whether by design or by chance, this dataset also contains some recordings exhibiting partial sensor detachment caused by intense motion. This is a type of failure that is likely to occur during FFs monitoring and therefore provides a valuable opportunity to test the algorithm's robustness.

While being extremely relevant, the original recordings do not provide information about the exact PPG or ACC integrated circuit (IC) used, nor about their specifications and configuration details.

The original format of the dataset was built around Matlab, but since then the data has been converted into more portable formats and is now available as comma-separated values (CSV) or time series files, making it easy to use in modern languages and environments.

This dataset is of particular interest for validating this thesis, as it contains a substantial amount of MA resulting from the subjects' treadmill exercises. These artefacts provide a reasonable proxy for the type of MA expected during real-world FFs vital-sign monitoring, where vigorous body movements and sensor displacement are likely to occur. Furthermore, the availability of the corresponding ACC measurements enables the use of MA reduction techniques that rely on ACC data.

Although the dataset was originally recorded using a wrist-worn device during treadmill exercises, it remains relevant because the resulting MAs arise from substantial body movements and sensor displacement. The motion artefacts encountered during FFs operations are expected to differ in their exact characteristics, such as amplitude, frequency content, and temporal patterns. Nevertheless, the dataset

provides a suitable basis for developing and evaluating the proposed algorithm. The algorithm can subsequently be refined and tuned using recordings acquired under FF-specific conditions.

Figure 1.7 shows the first 500 unprocessed samples of the first IEEEPPG training recording, sampled at 125 Hz.

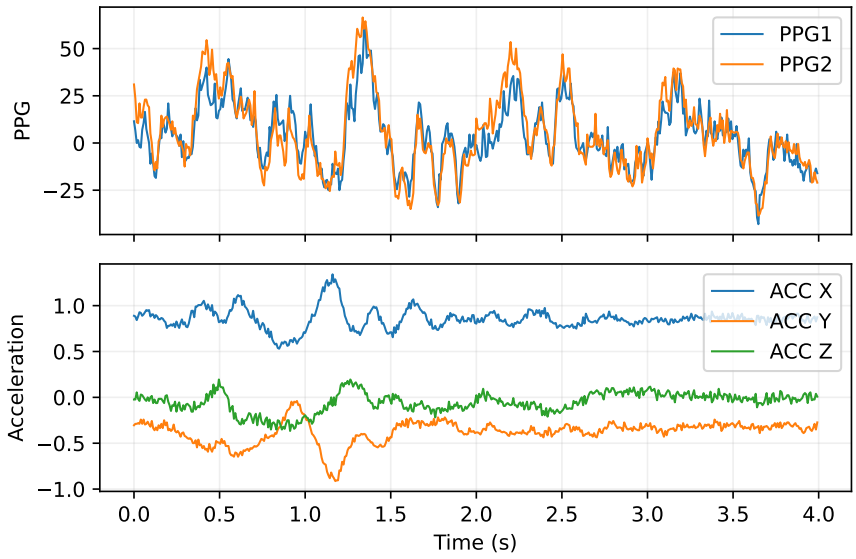


Figure 1.7: Example of IEEEPPG dataset recordings.

In this plot, the PPG traces are no longer as clean as in figure 1.2 or figure 1.5a. The MA amplitude is large enough that heartbeat peaks are difficult to visually identify in the time-domain. For reference, the corresponding HR in this segment is approximately 74 BPM, which corresponds to roughly 3.2 beats over the 4 s window shown.

1.6 State of the art

In 2017, Temko et al. proposed a spectrum-based algorithm to compensate the MAs in PPG signals and provide a more accurate HR estimation in their presence [19]. In addition to a PPG signal, their algorithm also uses an ACC to measure the intensity and spectrum of motion over a window, helping to identify the spectral bins that are likely to be affected by MAs and compensate for them. By doing so, they are able to overcome the limitations of simpler spectrum-based algorithms and provide a trustworthy HR estimation even in the presence of MAs with higher

magnitude than the HR signal. Their proposed solution also incorporates a phase vocoder to further refine the HR estimation beyond the spectral resolution and a user-adaptive post-processing step to track the subject's physiological state. They claim that this processing is lightweight enough to run in real-time (online), and they also propose a more computationally intensive offline version that adds a Viterbi decoding to further improve the estimation by taking into account the temporal evolution of the HR and MAs.

As it was already fairly well known dataset, they validated their algorithm on the 25 recordings from the IEEEPPG dataset. From this validation, they achieved a mean absolute error (MAE) of 1.97 BPM and 1.37 BPM for the online and offline versions of their algorithm respectively, which represents a significant improvement over simpler spectrum-based algorithms that lose accuracy as soon as the MAs magnitude exceeds the HR signal magnitude.

Their algorithm is structured as presented in figure 1.8. The rest of this subsection provides a brief description of each step and sub-steps as well as the rationale behind each of them when needed. More details on the algorithm can be found in the original paper [19].

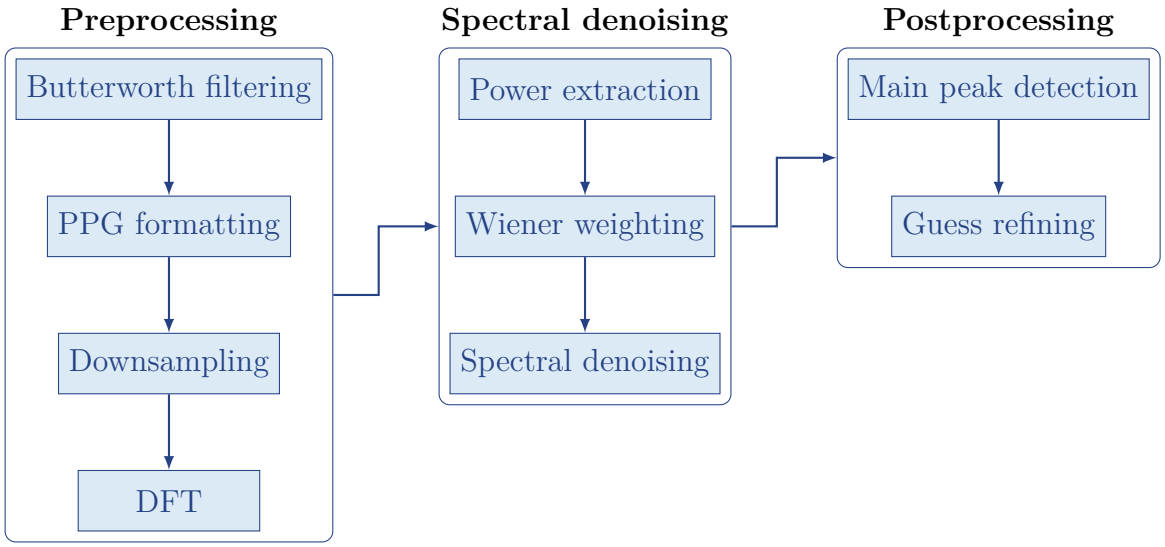


Figure 1.8: Temko et al. algorithm pipeline [19].

Their processing is performed on 8 seconds windows with a 2 seconds step, providing overlapping updates in the time-domain. This means that their algorithm could in theory produce an estimation every 2 s, which is a good compromise between responsiveness and accuracy, as shorter windows would lead to lower frequency

resolution and thus a less accurate estimations, while longer windows would lead to higher frequency resolution but also less responsive estimations. This parameter can be adjusted to fit the needs of the application, but it is important to keep in mind that it represents a trade-off between accuracy and responsiveness. Additionally, the use of overlapping windows allows for a smoother estimation over time, as it provides more frequent updates while still maintaining good frequency resolution.

Before performing any useful conversion, the two PPG channels and three ACC channels are preprocessed to precondition the signal and make them more suitable for the next steps. This begins with a Butterworth bandpass filter to remove low-frequency components and high-frequency noises. For this purpose, they use a 4^{th} order Butterworth filter with a passband of 0.4 Hz to 4 Hz, which corresponds to an HR ranging from 24 BPM to 240 BPM, a reasonable range for human HR [20].

Once filtered, the PPG signal are processed to make them more suitable for the next steps. This formatting consists of standardizing the signal to obtain zero mean and unit variance channels. Once the signals are standardized, the two PPG channels are averaged to get a single resulting channel.

After fusing the two PPG channels together, the resulting PPG channel, along with the three ACC channels are downsampled from the original sampling frequency of 125 Hz to 25 Hz. This is done to reduce the computational cost of the next steps, as the original sampling frequency had a Nyquist frequency of 62.5 Hz = 3750 BPM, which is an order of magnitude higher than the maximum expected human HR.

Finally, a DFT is performed on the four resulting channels to convert them from the time-domain into the frequency-domain. For their implementation, the authors used a 1024-point DFT, with zero padding applied before this DFT to obtain a better frequency resolution, with the important caveat that this zero padding "does not create new information but allows for a better revelation of the existing information in the signal" [19].

After this pre-processing step, one spectrum of the PPG signal and three spectra of the ACC signals are obtained. The following steps are where the innovation of Temko et al.'s algorithm lies, as they use the ACC spectra to identify the spectral bins that are likely to be affected by MAs and compensate for them in the PPG spectrum. The first step of this compensation is to convert each channel from a complex vector to a power spectrum by taking the square of the magnitude of each complex value. At this step, the squared values in the ACC channels are averaged to yield a single ACC power spectrum. To have a stable measurement range, these power spectra are then normalized by their mean value to keep all values within

$[0, 1]$ and remove the constant component.

With the PPG and ACC power spectra available, the spectral compensation step can now be formulated. As MA is additive noise, the power of the PPG spectrum can be approximated by equation (1.3), with $X(f)$ being the observed spectrum, $S(f)$ being the HR spectrum measured by the PPG and $N(f)$ being the additive MA spectrum. Rewriting this equation, equation (1.6) can be derived. By regrouping the terms multiplying $X(f)$, equation (1.7) is obtained, which corresponds to the original formulation of the compensation step in Temko et al.'s algorithm. In this formulation, $W(f)$ acts as a Wiener weighting vector on the observed spectrum $X(f)$.

$$X(f) = S(f) + N(f) \tag{1.3}$$
$$\tilde{S}(f) = X(f) - N(f) \tag{1.4}$$
$$= \left(1 - \frac{N(f)}{X(f)}\right) \cdot X(f) \tag{1.5}$$
$$= \frac{X(f) - N(f)}{X(f)} \cdot X(f) \tag{1.6}$$
$$= W(f) \cdot X(f) \tag{1.7}$$

Since motion can change at any moment, unlike the physiological HR signal, the HR and MA signals can reasonably be assumed to be uncorrelated. Swapping to the notation of power spectra $P_{\gamma\gamma}(f)$ representing the auto power spectral density (PSD) of signal γ, they obtain equation (1.8) and equation (1.10).

$$W(f) = 1 - \frac{P_{NN}(f)}{P_{XX}(f)} \tag{1.8}$$
$$= \frac{P_{XX}(f) - P_{NN}(f)}{P_{XX}(f)} \tag{1.9}$$
$$= \frac{P_{SS}(f)}{P_{SS}(f) + P_{NN}(f)} \tag{1.10}$$

By introducing memory or temporal smoothing into equation (1.8), they obtain the expression of the Wiener weights of the first filter as presented in equation (1.11). These weights only depend on present and C past values of the observed PSDs.

$$w_1(t, k) = 1 - \frac{P_{NN}(t, k)}{\frac{1}{C}\Sigma_{i=t-C+1}^{t} P_{XX}(i, k)} \tag{1.11}$$

The second filter coefficients are computed by applying the same logic to equation (1.10), which yields equation (1.13). In contrast with the w_1 coefficients, the w_2 coefficients also depend on their own past values, which means they have to be computed recursively starting from an initial value of w_2 for the first C windows. This recursive nature of the w_2 coefficients makes them more computationally intensive than the w_1 coefficients, but they also provide a better compensation as they take into account the temporal evolution of the HR and the MA.

$$P_{SS,2}(t,k) \approx \frac{1}{C}\Sigma_{i=t-C}^{t-1} w_2(i,k) P_{XX}(i,k) \tag{1.12}$$

$$w_2(t,k) = \frac{\frac{1}{C}\Sigma_{i=t-C}^{t-1} w_2(i,k) P_{XX}(i,k)}{\frac{1}{C}\Sigma_{i=t-C}^{t-1} w_2(i,k) P_{XX}(i,k) + P_{NN}(t,k)} \tag{1.13}$$

Here, t denotes the window index, and k denotes the index of the corresponding PSD bin. This averaging assumes that the HR varies slowly over C windows, while the MA contribution is more random and is therefore attenuated by averaging over different windows.

Applying these two Wiener filters to the observed PPG spectrum $P_{XX}(t,k)$, they obtain two compensated spectra $P_{SS,1}(t,k)$ and $P_{SS,2}(t,k)$, which are then scaled and averaged to produce a de-noised final spectral envelope. The scaling is performed using their standard deviation "because unlike w_2, w_1 can have negative values, when the scaled power of observed noise is larger than the scaled power of the observed signal for certain frequencies" [19]. The HR is then estimated by identifying the frequency bin with the highest power in this final spectral envelope, which should correspond to the HR frequency.

The authors then propose an optional phase vocoder step to further refine the HR estimation beyond the spectral resolution by taking advantage of the phase information in the signal and estimating its change over windows. For the sake of keeping this background chapter concise, the details of this step are not described here. Further details can be found in the original paper [19].

In equation (1.14), τ_i represents the search range's width at the current window i, $f(t)$ represents the estimated HR frequency at window t and D is a parameter that defines how many past windows are considered for the search range. The parameter D is noted k in the original paper, but it is renamed here to avoid confusion with the PSD bin index and ensure consistency in parameter naming. In later figures, the search bounds are represented as the labels "HR min" and "HR max" on the plots, which correspond to the minimum and maximum frequencies considered during the estimation of the HR at the current window.

$$\tau_i = \max\{|f(t) - f(t-1)| : D < t < i - 1\} \tag{1.14}$$

Finally, the post-processing step is reached, which has both an online lightweight version and a more computationally intensive offline version. As for the vocoder step, the details of the offline version are not described here. For the online version, it first consists of restricting the search range of the final PSD peak to be around the previous estimation by a maximum τ_i BPM, as in equation (1.14). After this restriction, if the current HR estimation is at more than 5 BPM away from the previous estimation, the final HR estimate is obtained by combining the spectrum's maximum value bin with a linear regression of the previous estimations as in equation (1.15). The paper uses $\alpha = 0.8$ to assign more weight to the current estimation while still taking into account past estimations to provide a smoother estimation over time.

$$\tilde{f}(i) = \alpha f(i) + (1 - \alpha) f_{\text{reg}}(i) \tag{1.15}$$

Table 1.1 shows the results of Temko et al.'s algorithm on the IEEEPPG dataset for different configurations obtained by enabling certain steps of the algorithm. The abbreviations are defined as follows:

- B stands for Baseline, which corresponds to all pre-processing steps and online post processing steps but without any routine to compensate for MAs.
- SS stands for Spectral Subtraction, which corresponds to using equation (1.11) and equation (1.13) with $C = 1$, meaning that no historical averaging is performed on the PSD values.
- $W1$ and $W2$ correspond to using equation (1.11) and equation (1.13) with $C = 15$, meaning that the PSD values are averaged over the past 15 windows to obtain a longer acquisition period to compute the Wiener weights.
- Wx corresponds to using both $W1$ and $W2$ and averaging their results as described previously.
- PV stands for Phase Vocoder, which corresponds to enabling the phase vocoder refinement step.

Configuration	B	B + SS	B + W1	B + W2	B + Wx	B + Wx + PV
MAE	13.11	5.08	4.19	3.30	2.21	1.97

Table 1.1: Results of Temko et al.'s algorithm on the IEEEPPG dataset [19].

As expected, the baseline configuration performs poorly with a MAE of 13.11 BPM. This score acts as a sanity check for the dataset, as it confirms that the presence of sufficient MAs in the recordings indeed causes a significant degradation of the estimation performance, which is consistent with the conclusions of Martin Brans' thesis [4].

Adding more and more steps to the algorithm allows the MAE to be significantly reduced down to 1.97 BPM with the full offline configuration, which is a significant improvement over the baseline and shows the effectiveness of the proposed techniques to compensate for MA.

At the end of their paper, Temko et al. also compare the number of parameters of their algorithm to that of other algorithms achieving similar performance on the same dataset, and they show that they only need two de-noising parameters (C and D) and one post-processing parameter (α). This significantly reduces the risk of poor parameter tuning and overall makes the algorithm easier to tune.

1.7 This thesis

The results in table 1.1 show that Temko et al.'s spectrum-based approach can achieve high accuracy on the IEEEPPG dataset, but their implementation is designed in Matlab and targets an offline research context rather than a real-time, edge-compatible system for FFs monitoring. In addition, the original pipeline offers limited flexibility to balance accuracy, responsiveness, and computational cost when motion conditions or deployment constraints change.

This thesis is therefore structured around two complementary steps aimed at bridging the gap between the state of the art and a deployment-ready solution.

First, it implements a dedicated testbench for an extended version of Temko et al.'s algorithm, adding configurable features and parameters that enable systematic evaluation and tuning under different conditions.

Second, it develops a physical prototype and deploys the algorithm on a real-time edge device, enabling end-to-end validation on actual hardware and revealing practical issues that are invisible in purely offline evaluations.

Chapter 2

Heart rate monitoring algorithm

This chapter describes the design, implementation and performance evaluation of the HR monitoring algorithm developed in this thesis.

2.1 Design choices

The algorithm developed in this thesis is heavily based on the paper by Temko et al. [19] presented in section 1.6. Building on this paper, the algorithm adds configurable parameters and features to better fit specific use cases.

Figure 2.1 shows the block diagram of the algorithm developed in this thesis. The following subsections describe the modifications made to the original Temko et al. algorithm.

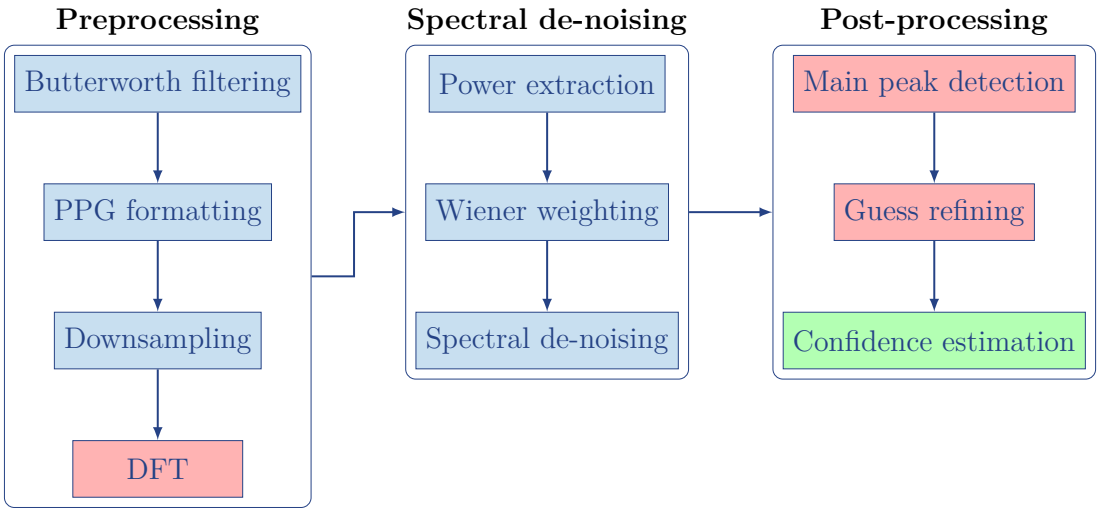


Figure 2.1: Pipeline of the proposed algorithm. ■ boxes indicate unchanged steps, ■ boxes indicate improved steps, and ■ boxes indicate new steps.

The general structure and signal processing pipeline of the algorithm remain unchanged, with some modifications or additions where relevant to the specific use case.

2.1.1 Preprocessing and windowing

The first modification is the application of a Hann window directly to the input signal in the time domain, before computing the DFT to reduce spectral leakage caused by possible sharp transitions at the edges of the window. Each time-domain sample is multiplied by the corresponding Hann window coefficient, such that the windowed signal is given by $x_{\text{windowed}}[n] = x[n] \cdot w[n]$. This is a common practice in signal processing to improve the quality of the spectral transformation, and is necessary because the DFT expects a periodic signal as input, and providing a non-periodic signal would introduce artifacts. This windowing is a type of raised-cosine window [21] and forces both the oldest and newest values to zero, as can be seen in figure 2.2, where the DFT size is $N_{\text{DFT}} = 1024$.

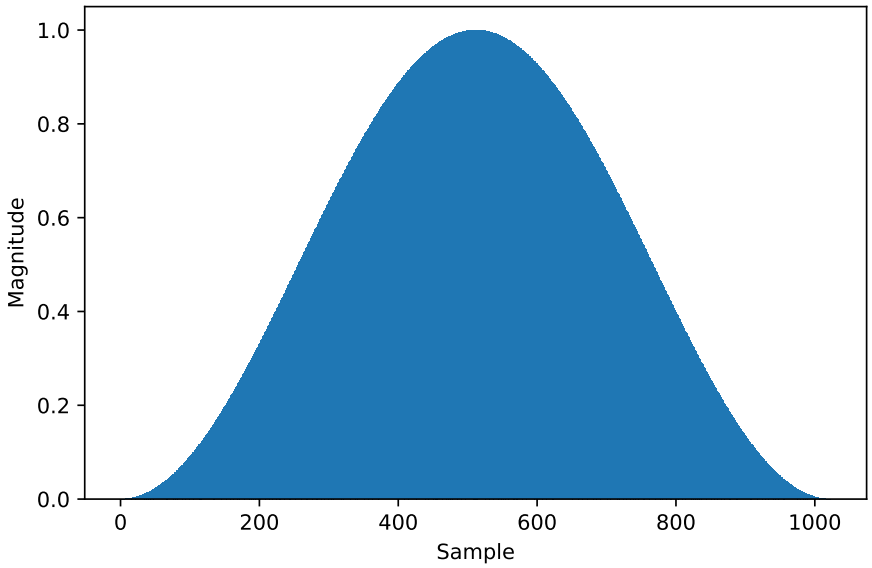


Figure 2.2: Hann window coefficients.

While this common filtering step is often implicitly assumed in many algorithms, it is mentioned neither in their paper [19] nor in the Matlab code provided by Temko et al. This omission may have been intended to keep the algorithm as lightweight as possible, but since then the computational cost of windowing has become negligible compared to that of the rest of the algorithm, especially with hardware accelerations and vectorization. Therefore, there is virtually no downside to adding it. All other parameters of this step, such as the number of DFT points and the use of zero-padding, remain unchanged from the original algorithm.

2.1.2 Reworked search range

The original Temko et al. algorithm makes the reasonable assumption that the HR does not change significantly between two consecutive windows and therefore uses the previous HR estimation to limit the search space, considering only DFT bins within a given radius around the previous estimate. Reconsidering the nature of this search range, it can also be seen as multiplying the DFT spectrum by a rectangular window centered around the previous estimate. To increase the configurability of the algorithm, the search range is now defined as a shaping function that takes the DFT spectrum as input and returns a weighted spectrum with bounds as an output. This makes it possible to implement more complex search ranges and post-DFT processing to improve the detection of the HR peak in the spectrum.

Some continuous shaping functions can be seen in figure 2.3. A smooth shaping

function ensures a gradual transition from the plateau of the search range to the boundaries, whereas a linear shaping function does not provide such a gradual transition.

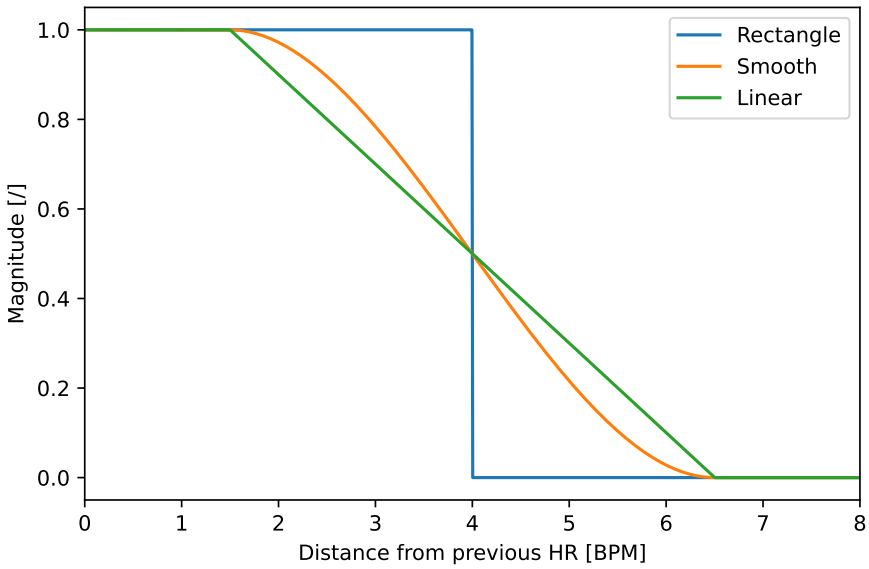


Figure 2.3: Examples of shaping functions.

As the shaping fonctions are sampled and discretized when used in the algorithm, this smoothness property would be lost anyway. Therefore, the linear shaping function is preferred due to its lower computational cost. The rectangular shaping function corresponds to the range restriction in the original Temko et al. algorithm, and is retained as an option for testing purposes.

For any shaping function, the search range is not reduced until at least D HR estimates are available in memory in order to avoid focusing on an inaccurate estimation before the algorithm's noise cancellation has had time to stabilize. To make this search range adaptive, its width is also linked to the confidence in the estimation, reducing the computational cost when the estimate is highly reliable and expanding the search range when the estimation is uncertain. More details on how the confidence score is computed are provided in section 2.1.3.

Using the search range definition given in equation (1.14) leads to undesirable behavior when the HR remains stable over several consecutive windows. In this situation, the $|f(t) - f(t-1)|$ term approaches zero, causing the search range to collapse to a single bin. After the activity resumes, the HR may change by more than this small delta between two consecutive windows, causing the algorithm to lose track of it.

To avoid this issue, a minimum search radius $SR_\Delta = 5$ BPM is added to the search range definition as shown in equation (2.1).

$$\tau_i = \max\{|f(t) - f(t-1)| + SR_\Delta : D < t < i - 1\} \tag{2.1}$$

This lower bound is applied regardless of the chosen shaping function, ensuring that a minimum search range is maintained even when the HR is stable and the estimation confidence is at its maximum. This allows the algorithm to better track sudden changes in HR, at the cost of a slightl increase in computational cost when the HR is stable.

During the guess refinement step, the original equation and underlying reasoning remain unchanged. The only modification from the original algorithm is the addition of easily configurable parameters α and a smoothing threshold, allowing for more or less aggressive smoothing of the HR estimation.

2.1.3 Confidence estimation

Finally, a confidence block is added at the end of the algorithm to provide the end user with a confidence level for each HR estimation. This confidence level is based on both the linearity of the newest prediction and the spectral concentration around the estimated HR. This confidence level is useful for quickly assessing the quality of an estimate without having to manually inspect each individual spectral content. This level provides an immediate indication of whether the estimate is reliable or should be treated with caution.

This confidence level is computed as a weighted sum of a deviation score S_{dev} and a spectrum shape score S_{diss} as defined in equation (2.2). Each subscore is constrained in the $[0, 1]$ interval to ensure a normalized output, provided that w_{dev} and w_{diss} sum up to 1.

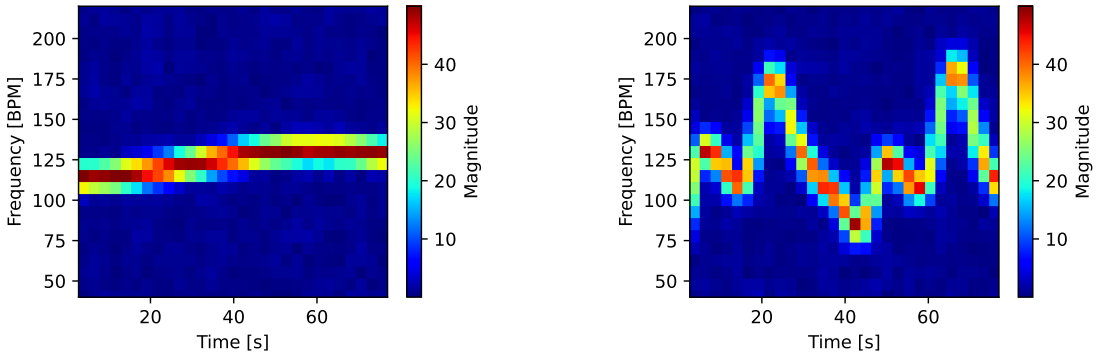
$$\mathrm{S}_{\text{conf}} = w_{\text{dev}}\, S_{\text{dev}} + w_{\text{diss}}\, S_{\text{diss}} \tag{2.2}$$

The deviation score detailed in equation (2.3) is based on the root mean square error (RMSE) of a linear regression applied to the most recent HR estimates, and provides a good indication of the stable progression of the HR.

$$
\begin{aligned}
a &= \frac{n\sum_{i=0}^{D-1} i \cdot f_{HR}(i) - (\sum_{i=0}^{D-1} i)(\sum_{i=0}^{D-1} f_{HR}(i))}{D\sum_{i=0}^{D-1} i^2 - (\sum_{i=0}^{D-1} i)^2} \\
b &= \frac{\sum_{i=0}^{D-1} f_{HR}(i) - a\sum_{i=0}^{D-1} i}{D} \\
\text{dev} &= \sqrt{\frac{1}{D}\sum_{i=0}^{D-1}\left(f_{HR}(i) - (a \cdot i + b)\right)^2} \\
S_{\text{dev}} &= \text{clamp}\left(1 - \frac{\text{dev}}{\text{dev}_{\max}}, 0, 1\right)
\end{aligned}
\tag{2.3}
$$

A low deviation score does not necessarily imply an inaccurate estimate, as the HR can change its trend rapidly under high stress conditions. Despite this, this score is still relevant as a high deviation score is still a good indicator that the estimates follow a linear trend and can therefore be considered reliable. As this component of the confidence score is linked to the temporal evolution of the estimated HR relative to time, it cannot be computed during the first few windows after device initialization and therefore requires a warm-up period.

Figure 2.4 presents a visual representation of respectively a high and a low deviation score. In figure 2.4a, the spectrum maxima follow a slowly varying linear trend, whereas in figure 2.4b, they exhibit a more erratic distribution.



(a) High deviation score.

(b) Low deviation score.

Figure 2.4: Deviation score examples.

The parameter $\text{dev}_{\max}$ can be tuned to better reflect the maximum plausible change in a FF's HR over time, ensuring that this score remains relevant with different cardiac profiles.

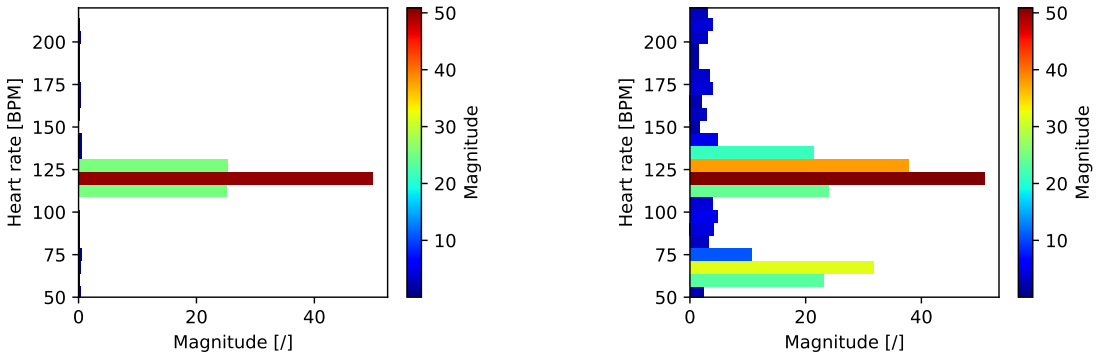
To provide a more comprehensive assessment of the confidence in an estimate,

the spectral dissipation is also incorporated into the overall confidence score. The spectrum dissipation score defined in equation (2.4) is related to the distribution of energy within the spectrum after Wiener filtering. It is computed as the sum of the entire spectrum weighted by a Gaussian function centered on the estimated HR, giving greater importance to nearby bins and divided by the total energy for normalization.

$$
\begin{aligned}
w_k &= \exp\left(-\frac{(k - \mathrm{idx}(f_{\mathrm{HR}}))^2}{2\sigma_{\mathrm{diss}}^2}\right) \\
S_{\mathrm{diss}} &= \mathrm{clamp}\left(\frac{\kappa}{L}\sum_{i=0}^{L-1}\frac{\sum_k w_k\, P_{SS}(i,k)}{\sum_k P_{SS}(i,k)},\, 0,\, 1\right)
\end{aligned}
\tag{2.4}
$$

This score provides a good indication of the clarity of the HR peak in the spectrum, and consequently of the effectiveness of MA filtering and the quality of the estimate. Since this component of the confidence score component relies only on the contained within a single DFT window, it does not require a warm-up period and can be computed for any window unlike the deviation score. The score is averaged over up to (L) windows, or over all available windows when fewer than L windows are present, and can therefore be computed as soon as at least one DFT window is available.

Figure 2.5 shows examples of DFT outputs resulting in high and low dissipation scores. The closely grouped bins in figure 2.5a indicate an effective noise cancellation, whereas the more dispersed bins in figure 2.5b are witness that the noise cancellation did not perform as well.



(a) High dissipation score.

(b) Low dissipation score.

Figure 2.5: Dissipation score examples.

The parameter σ_{diss} controls the sensitivity to spread in the spectrum, and is tuned empirically using a fitting dataset.

2.1.4 Parameter tuning

With these additions, the algorithm's data processing pipeline now exposes a broader range of tuning parameters. They control how aggressively MAs are canceled, the width of the search region around the previous HR estimate, the strength of the post-processing smoothing and the conservatism of the confidence score when assessing the reliability of an estimate. Their values can be optimized for specific use cases, and they can have a significant impact on both tracking performance and computational cost.

The following list summarizes the main parameters of the algorithm and their impact on performance and computational cost:

- C : the number of previous spectral windows considered during the Wiener filtering step. A higher C grants better cancellation of rappidly changing MAs, whereas a smaller C reduces computational costs and allows to have a smaller memory footprint.

- N_{DFT} : the size of the DFT window. A larger N_{DFT} can provide better frequency resolution but at the cost of increased computational complexity and latency, while a smaller N_{DFT} can reduce computational load but may lead to poorer frequency resolution.

- SR_{Δ} : the minimum search radius added to the search range definition around the previous HR estimate. A larger SR_{Δ} improves the tracking of sudden changes in HR at the cost of a slightly larger search space and increased computational cost, while a smaller value can make the algorithm too restrictive when the estimates stabilizes.

- D : the number of previous HR estimates considered for the linearization in the post-processing step. A value of D that is too large can negatively affect the tracking by including older estimates that are no longer relevant, whereas a value that is too small can make the impact of the linearization negligible. This parameter also determines the number of previous estimates to consider when computing the confidence score.

- α : the extent to which the linearization affects the estimation when it is applied. A value of α that is too high can make the linearization act too aggressively and make the estimation overly smooth, while a value that is too low can make its effect on the estimation negligible.

- β : the threshold used to apply the linearization in the post-processing step. A value of β that is too high can prevent the linearization from ever being applied, while a value that is too low can make it trigger too often and cause the linearization to act as a low-pass filter.

- κ : the gain of the spectrum dissipation score. A value of κ that is too high can make the spectrum dissipation score swing too agressively from minimum to maximum, while a value that is too small may prevent this confidence score metric from reaching its maximum.
- dev_{max} : the deviation normalizer for the deviation score. A value of dev_{max} that is too large can make the deviation score insensitive to the distance from the linear prediction, while a value that is too small can make it too sensitive.
- w_{dev} : the weight of the deviation-based confidence score in the combined confidence metric. A larger w_{dev} gives more importance to the linearity of the recent HR estimates, while a smaller value makes the final confidence more dependent on the spectrum dissipation score.
- σ_{diss} : the standard deviation of the Gaussian function used to weight the spectrum in the dissipation score. A value of σ_{diss} that is too large can make the dissipation score insensitive to spectral spread, while a value that is too small can make it overly sensitive to spectral spread.
- w_{diss} : the weight of the spectrum dissipation-based confidence score in the combined confidence metric. A larger w_{diss} makes the confidence metric more sensitive to the spectral concentration around the estimated HR, while a smaller value makes it less sensitive to this aspect.
- L : the maximum number of past windows over which to average the dissipation score. A larger L results in stronger smoothing, causing the score to respond more slowly to changes, while a smaller L results in a more responsive but potentially more erratic score.

2.1.5 Changes summary

The modifications presented in this section represent both adaptations to optimize the algorithm for specific use cases and improvements based on signal processing best practices.

The addition of windowing before the DFT is a well-established technique that was omitted from the original Temko et al. implementation, likely to reduce complexity, but is now included due to its negligible computational cost and significant benefit in reducing spectral leakage.

The reworked search range, parameterized as a generalized shaping function, improves configurability and adaptability to different cardiac conditions while maintaining compatibility with the original rectangular search window.

The introduction of a confidence score provides end users with an immediate assessment of estimation reliability without requiring manual inspection of

spectral data, adding practical value to the algorithm's deployment in real-world applications.

Finally, the exposure of tuning parameters such as C, N_{DFT}, and α enables optimization for specific scenarios, balancing tracking performance, computational cost, and latency according to the requirements of the target platform or use case.

In summary, these changes enhance the algorithm's robustness, configurability, and practical usability while preserving the core design principles of the original work.

2.2 Implementation

Once the design of the theoretical algorithm is completed, the next step is to implement it as working software. This section describes the implementation of the algorithm and the choices made during this process.

2.2.1 Language choice

In order to convert the mathematical formulation of the algorithm into a working implementation, the first step is to choose an appropriate programming language.

For this choice, several factors were considered to ensure that the implementation meets the requirements of the algorithm and those of the target platform.

First, the algorithm is intended to be used on an embedded platform, so a low-level capable language able of producing efficient binaries is essential.

Second, the algorithm is intended to be used in a real-time context, so a language with good support for concurrency and parallelism is necessary.

Third, the algorithm is not as trivial as simply implementing a mathematical formula, so a language with a higher level of abstraction and good support for higher-order primitives is preferred.

Finally, since the algorithm is intended for real-time context use, it needs to be implemented in a language that eases the development of safe and reliable code, with good support for validation, testing and debugging.

Historically, the most common and widely used languages for embedded development are C and C++. These languages are as low-level as reasonable and allow one to produce very efficient code after compiler optimizations. However, since these languages allow one to manually manage memory and usually require the use of references by pointers, they are prone to memory safety issues and undefined behavior. C++ has some features to mitigate these issues, but as the creator of

C++ Bjarne Stroustrup said, “C makes it easy to shoot yourself in the foot; C++ makes it harder, but when you do, it blows your whole leg off” [22].

Building on the struggles associated with these issues and the feedbacks gathered over the last decades, a new language called Rust has emerged as a promising alternative to C and C++. Rust is a language with a higher level of abstraction that aims to provide the same level of performance and low-level control as C and C++ when needed, while also providing strong guarantees of memory and thread safety, even going as far as preventing data races at compile time. Rust achieves this by using a strict ownership model and a borrow checker that enforces rules on how references to data can be used. This allows developers to write safe and efficient code without having to worry about common pitfalls like dangling pointers, null references, or data races. The downside of this compile-time checking system is that designing, compiling and shipping Rust code may take more time than with C and C++, due to the added complexity of writing code that meets the strict safety requirements imposed by the language's compiler.

Table 2.1 summarizes the comparison of the three candidate languages considered for the implementation of the algorithm, based on the criteria mentioned above. A checkmark indicates that the language provides the criterion directly or with strong language-level support, while an empty cell indicates that achieving the criterion generally requires additional conventions, libraries, or programmer discipline.

Criteria	C	C++	Rust
Low-level control	✓	✓	✓
Performance	✓	✓	✓
Memory safety			✓
Concurrency safety			✓
High-level abstractions		✓	✓

Table 2.1: Comparison of implementation language candidates.

For these reasons, Rust appears to be a well-suited choice as the implementation language for the algorithm. It provides a good balance between low-level control, performance, and safety, making it a suitable choice for implementing a real-time HR monitoring algorithm on an embedded platform.

Additionally, Rust has the capability to run C and C++ code via the `bindgen` tool, which allows developers to generate Rust bindings for C and C++ libraries. This makes it possible to leverage existing libraries and code written in C and

C++. However, interactions with foreign code require particular care, as the safety guarantees provided by Rust do not automatically extend to code written outside of Rust. For semantics and examples of the Rust programming language, please refer to the official Rust documentation [23].

2.2.2 Estimation within a simulator

To run on an embedded platform with limited resources, the algorithm needs to remove as many unnecessary dependencies as possible. To achieve this, Rust provides an attribute called `no_std` that allows Rust to be compiled without the standard library, which is usually not needed in an embedded context. This reduces the size of the compiled binary and avoids using resources for non-essential features.

However, prohibiting access to the standard library also means that we no longer have the option to use some of the higher-level abstractions provided by Rust's standard library, such as filesystem access, file parsing and GUI libraries. This is not an issue while the algorithm is running on an embedded platform, but it also means that it cannot take its input from a database of sensor data or from a file containing a recording of a sensor signal. Since it is important to validate this algorithm against the same dataset as the original Temko et al. algorithm to ensure a fair comparison of the performance, our algorithm also needs to be able to run in a desktop environment with access to the standard library to parse the input data file.

To address the need for both embedded deployment and desktop-based validation, the HR algorithm is implemented in the HeartTrack crate that enforces `no_std` by default. Having this crate without any `std` features ensures that the core algorithm remains lightweight and compatible with resource-constrained platforms. A separate `std` Simulator crate is then built around this common algorithm codebase. With access to the standard library, it provides the additional functionalities requiring a desktop environment, including the conversion between IEEEPPG's format and the input format expected by the algorithm. The simulator reads recorded sensor data from input files, converts each data segment into the format required by the HeartTrack estimator, and sequentially feeds these segments to the estimator in the same way as EmberGuard would provide live sensor measurements. The resulting HR estimates can then be compared against the reference values from the dataset, allowing the algorithm's performance to be evaluated and validated. The inner workings of EmberGuard are discussed in chapter 3.

As a result, the exact same algorithm implementation can be executed both in simulation, with access to the standard library and recorded sensor data, and on the embedded EmberGuard platform with real sensor inputs and limited resources.

Figure 2.6 illustrates the relationships between these executors, the HR algorithm and input or output data. The codebase written entirely in Rust is shown in ♦ orange/red, the codebase written in Python is shown in ♦ blue, and the files or streams are represented in ♦ green. The ■ Teal and the ■ purple dashed zones respectively represent the simulator and the EmberGuard platforms. The data used by the plotter can be either the logs of the HR estimator, or the logs of EmberGuard.

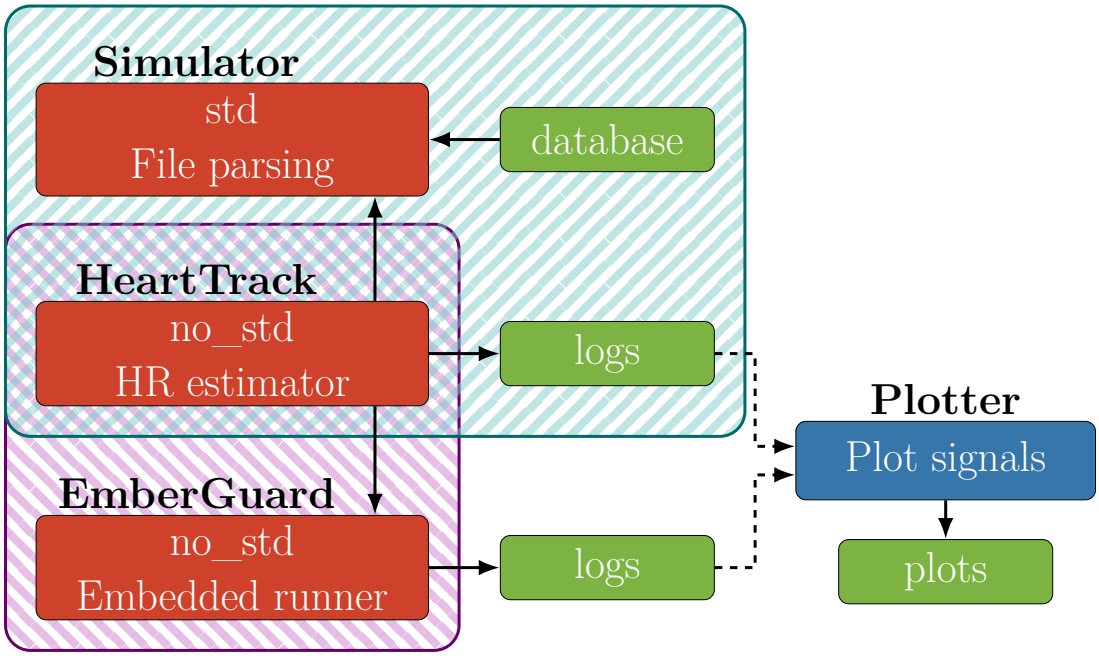


Figure 2.6: Block diagram of the algorithm validation framework.

2.2.3 Estimator's features

Features in Rust are a way to conditionally compile code based on the presence or absence of certain features, which are declared at compile time. A more in-depth explanation and examples can be found in [23].

Since most of the data to be logged from the HR algorithm are stored as private internal variables, the simulator does not have access to them and is therefore unable to log them to a file. To solve this issue, the HR algorithm provides a feature called `full-log` that adds a `std` logger to the otherwise `no_std` HeartTrack crate. This logger is then used to log these private fields of the algorithm to a file, with the path to the log file being passed as a parameter to the estimator upon initialization.

2.3 Performance metrics

Once the algorithm design is finalized and implemented in a ready-to-test software, the next step is to evaluate its performance. This evaluation is done by running HeartTrack within the simulator, feeding it diverse sensor datasets and comparing the estimated HRs with the reference values provided by the datasets.

First, the performance of the algorithm is evaluated using synthetic data generated by a custom signal generator.

Second, the performance of the algorithm is evaluated using the IEEEPPG dataset, which contains real-world sensor data collected from multiple subjects under various conditions.

2.3.1 Synthetic data performance evaluation

As a baseline for the algorithm's performance evaluation, a generator is used to create synthetic data with known HRs and MAs, allowing the algorithm's ability to accurately estimate the HR under controlled conditions to be validated. This also allows edge cases to be identified. From this generator, a dataset with a similar structure that of the IEEEPPG dataset is created, with custom HR and MA expressions over time.

Figure 2.7 shows the time-domain representation of the synthetic data generated by the proposed generator using equation (2.7) and equation (2.8).

In this example, the HR is set to 180 BPM, while the MA frequency is set to 80 BPM. Both the PPG and ACC signals are corrupted by independent Gaussian noise terms $\eta_{\text{PPG}}(t)$ and $\eta_{\text{ACC}}(t)$, each following a normal distribution with zero mean and a standard deviation of 0.05^2. In addition, all channels share the same MAs defined by equation (2.6). Unlike a stationary sinusoid, this artifact exhibits slow variations in both its instantaneous frequency and amplitude, making it more representative of real-world motion disturbances.

The term $e(t)$ is a slowly varying amplitude envelope generated from smoothed random noise. Together with the slowly varying instantaneous frequency $f_{\text{MA}}(t)$, it produces a MA whose amplitude and frequency evolve over time, resulting in a more realistic disturbance than a stationary sinusoid.

The coefficients d_j project the common MA onto the three orthogonal ACC axes. They correspond to the components of the unit vector

$$\mathbf{d} = \frac{(0.72,\ 0.48,\ 0.50)^\top}{\sqrt{0.72^2 + 0.48^2 + 0.50^2}}$$

and likewise, the matrix $\mathbf{b}$ represents the orientation of the gravity vector, with the X axis pointing upward in this example.

The multiplicative term A_i accounts for the channel-dependent optical response of the PPG sensors. Since each channel corresponds to a different wavelength, the light penetration depth and the interaction with the underlying tissue vary across channels. Consequently, each wavelength exhibits a different sensitivity to both pulsatile blood volume changes and motion-induced variations. Therefore, A_i introduces a channel-specific multiplicative bias affecting both the HR and the MA components of the PPG signals. The generated PPG signals are zero-centered for simplicity, as the algorithm discards the constant component during the preprocessing step.

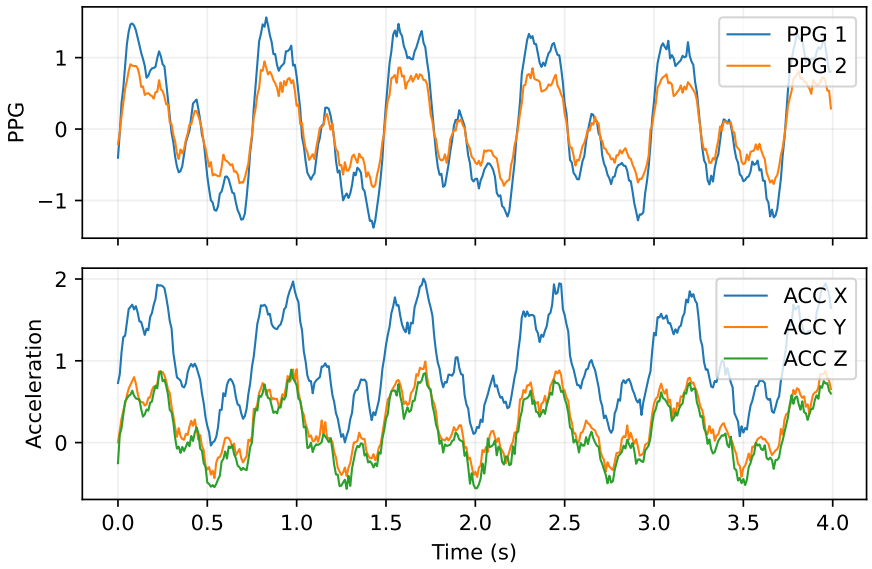


Figure 2.7: Synthetic data created by the generator.

$$\phi_{\mathrm{MA}}(t) = 2\pi \int_0^t f_{\mathrm{MA}}(\tau)\,\mathrm{d}\tau \tag{2.5}$$

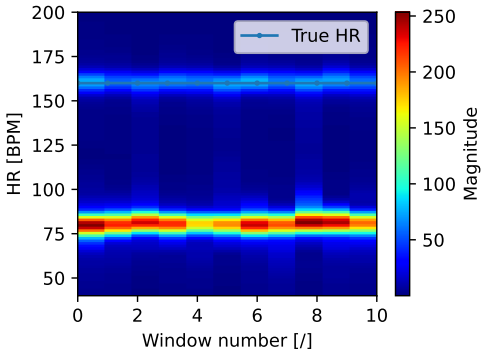
$$a(t) = e(t)\left[\sin(\phi_{\mathrm{MA}}(t)) + 0.1\sin(2\phi_{\mathrm{MA}}(t) + \varphi_2) + 0.5\sin(4\phi_{\mathrm{MA}}(t) + \varphi_4)\right] \tag{2.6}$$

$$\mathrm{PPG}_i(t) = A_i(0.5\sin(\phi_{\mathrm{HR}}(t)) + a(t)) + \eta_{\mathrm{PPG}}(t) \qquad i \in \{1, 2\}, \tag{2.7}$$

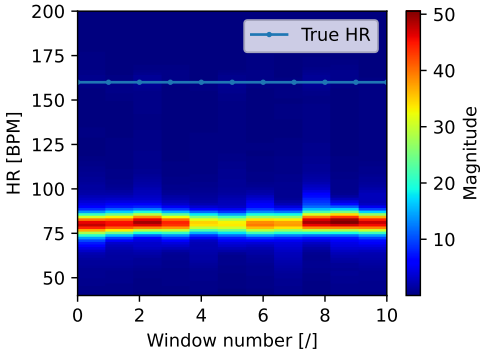
$$\mathrm{ACC}_j(t) = +b_j + d_j\, a(t) + \eta_{\mathrm{ACC}}(t) \qquad j \in \{x, y, z\}, \tag{2.8}$$

Although the generator is configured with higher MA amplitudes for this example, the generated synthetic data still exhibits a trend comparable to that of the IEEEPPG dataset, as illustrated in figure 1.7.

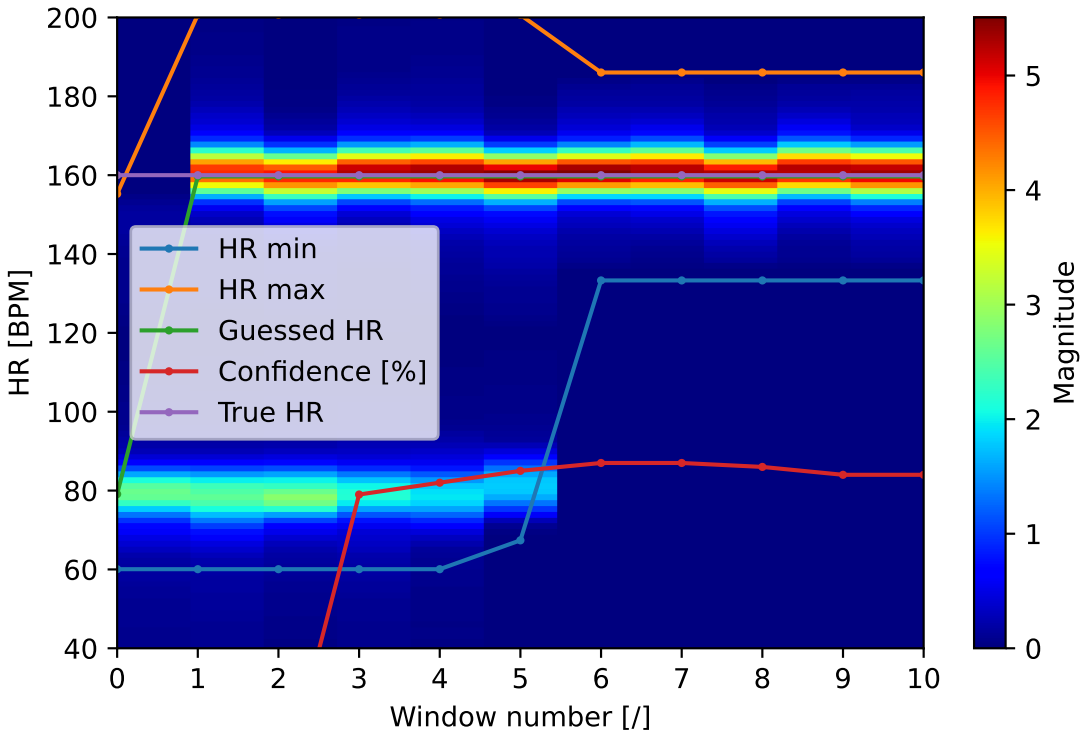
Running the algorithm on these synthetic signals produces the results shown in figure 2.8. For this first test, the generator is configured with $f_{\mathrm{MA}} = 80$ BPM and $f_{\mathrm{HR}} = 160$ BPM as in figure 2.7.



(a) PXX of the synthetic data.



(b) PNN of the synthetic data.



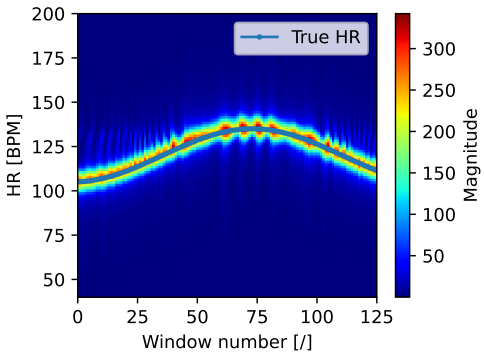
(c) PSS of the synthetic data.

Figure 2.8: Algorithm's spectra for synthetic data.

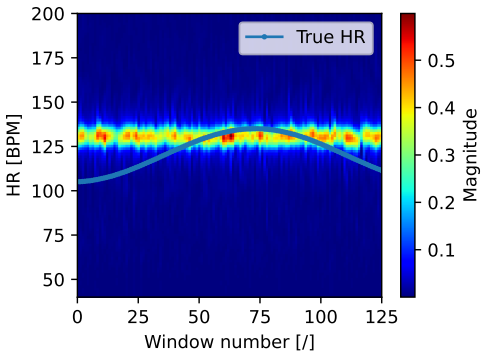
Once the history compensation has finished warming-up, the algorithm is able to cancel out the MA and track the HR accurately even when the signal-to-noise ratio (SNR) is low, as can be seen in figure 2.8c. This subfigure also shows the search bounds getting closer and the confidence rising after the warm-up period.

During the first 10 frames, the confidence does not yet reach its stable value. After 50 windows, it eventually reaches a stable value of over 90%.

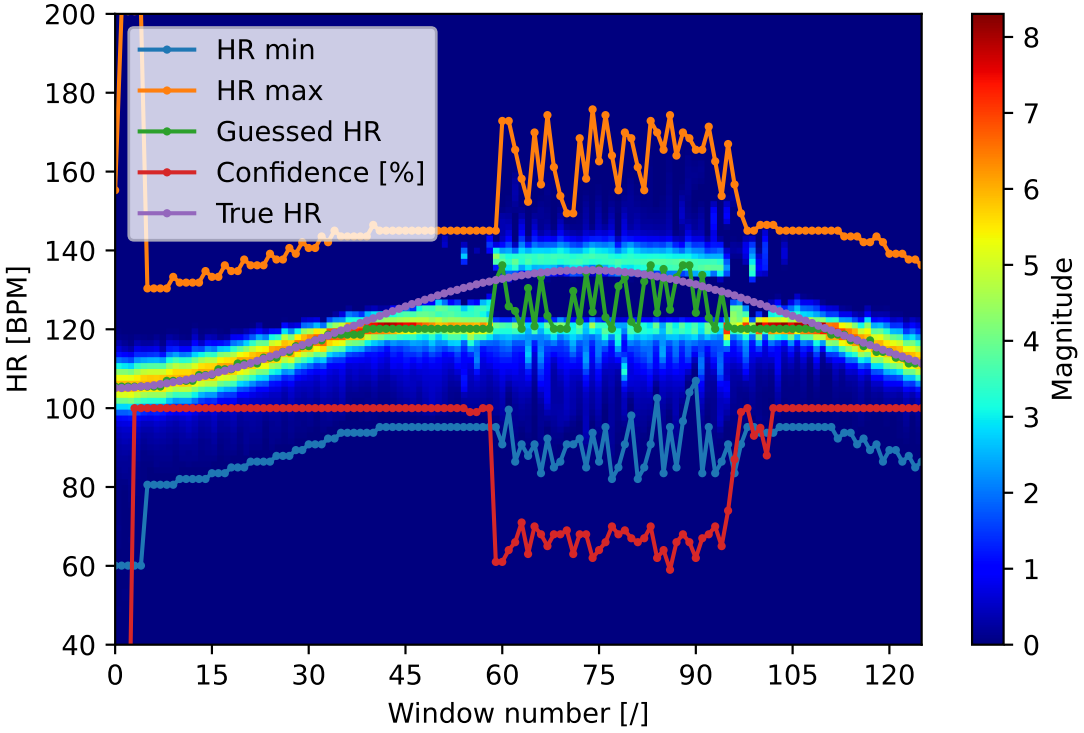
Taking a more complex example over a longer time span, figure 2.9 illustrates the behavior of the algorithm when the HR and MA frequencies become close. The generator is configured with $f_{\text{MA}} = 140$ BPM, while f_{HR} varies around 130 BPM.



(a) PXX of the synthetic data.



(b) PNN of the synthetic data.



(c) PSS of the synthetic data.

Figure 2.9: Spectrograms of the algorithm's input for the synthetic data.

The algorithm successfully tracks the HR when the two frequency components are sufficiently separated. However, when their frequencies converge, the MA

cancellation step suppresses the ACC frequency bins containing the HR component, resulting in a loss of tracking accuracy. The HR can only be recovered once the frequency separation becomes sufficient again.

Since a jump in the estimated frequency and a dilatation of the spectrum energy happen, the confidence level drops and only rises again after the issue is resolved. During this drop, the confidence score signals to the user that these measurements may not be accurate and should be interpreted with caution. This is exactly the use case the confidence score was designed for, and provides a reliable indicator of the quality of the estimate.

While the last example seems to highlight a major limitation of the algorithm, it still performs reasonably well, as shown in table 2.2. This is because if the assumption of a stationary dominant MA frequency is removed, a useful signal bypass such as this only occurs on at most a few windows at most, after which the algorithm is able to recover the HR estimate.

Example	MAE	Confidence [%]
Low SNR	0.73	96
Collision	2.36	96

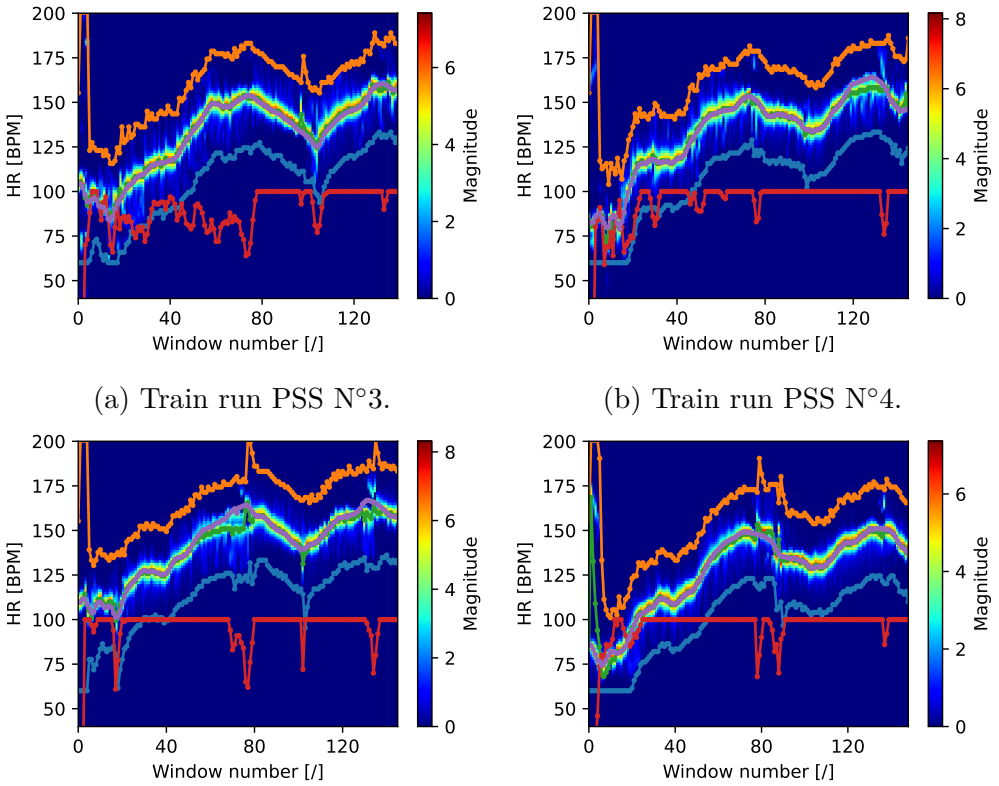
Table 2.2: Performance metrics for the synthetic data examples.

2.3.2 Real data performance evaluation

As mentioned previously, this algorithm is heavily based on the approach proposed by Temko et al. Therefore, the performance metrics and dataset used for its evaluation are the same as those employed in the original paper, ensuring a fair comparison between the two methods.

In the following spectrograms, the legend is omitted to preserve readability, but the same color scheme is used as in other PSS spectrograms, such as figure 2.9.

Evaluating the algorithm on the IEEEPPG dataset results in accurate HR estimates for most runs, as illustrated in figure 2.10. The algorithm is able to keep its guesses close to the ECG ground truth.



(a) Train run PSS N°3.

(b) Train run PSS N°4.

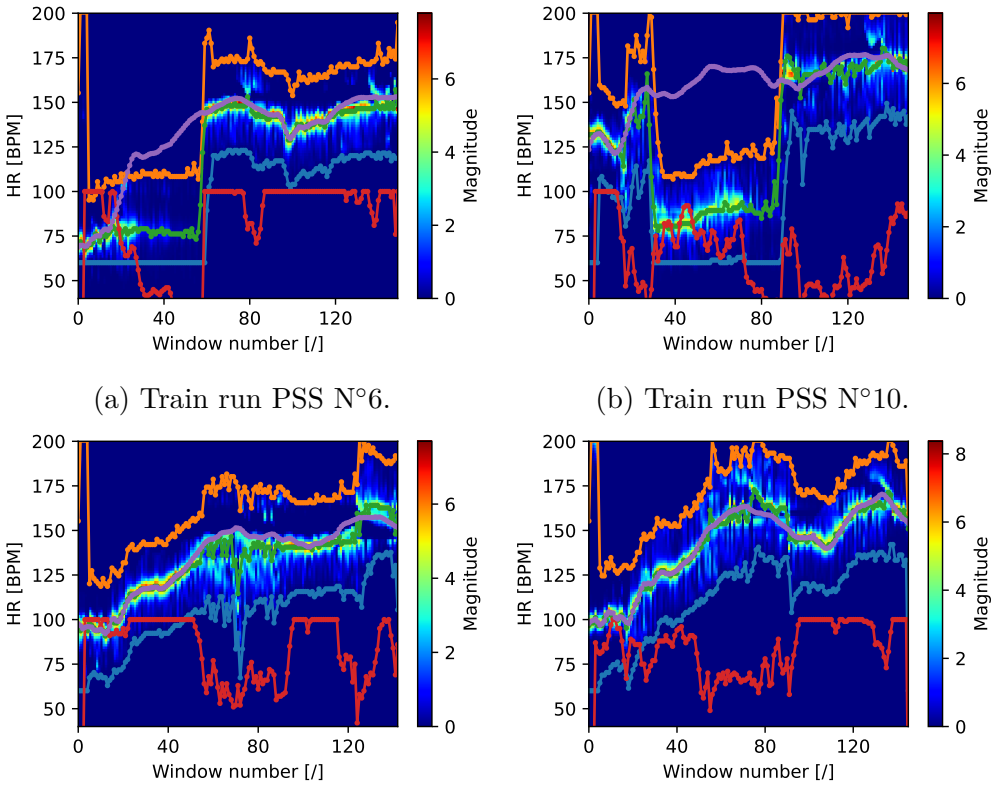
(c) Train run PSS N°5.

(d) Train run PSS N°9.

Figure 2.10: Examples of accurate estimations.

Although heuristic based, the confidence score reflects the quality of these estimations, with the exception of figure 2.10a for which the MA profile changes abruptly around the 75^{th} window, causing some MA energy to leak into the processed PSS spectrum and thereby lowering the dissipation score as defined in equation (2.4). This leads to a drop in confidence, even though the estimations are still within 5 BPM of the true value.

Some runs, however, result in estimation errors. Figure 2.11 shows examples of these events. All following explanations are based on the analysis of the PSS, PNN and PXX spectrograms, as well as on the insights gained from the synthetic data performance evaluation in section 2.3.1.



(a) Train run PSS N°6.

(b) Train run PSS N°10.

(c) Train run PSS N°7.

(d) Train run PSS N°12.

Figure 2.11: Examples of estimation errors.

In figure 2.11a and figure 2.11b, there may have been a loss of contact between the PPG sensor and the test subject around windows N°20 and N°25 respectively. These poor measurement conditions may have caused a sudden drop in the PPG signal amplitude, which in turn caused the algorithm to focus on the dominant MA frequency that had not yet been canceled by the Wiener filtering step, as this event occurred before the historical-based compensation had fully initialized. In figure 2.11a, once these local maxima are canceled, the algorithm's search range widens and the true HR frequency is recovered. As for figure 2.11b, the dominant MA frequency shifts down to 80 BPM just as the artifacts at 130 BPM are canceled, causing the algorithm to incorrectly track the new dominant MA frequency instead of the true HR. The algorithm is then able to recover once this dominant MA frequency is canceled. This second case proves to be extremely sensitive to the parameter values, with even slight variations leading to significant differences in the HR tracking of this record. This sensitivity, combined with the weak energy content of the PSS even after filtering, suggests that the sensor may have been misaligned for the remainder of the run following the initial loss of contact.

In figure 2.11c and figure 2.11d, the HR and MA frequency components overlap around windows N°50. As discussed for figure 2.9, this leads to a temporary overcorrection of the MAs and a temporary loss of HR signal tracking. The algorithm recovers well after these events, but the MAE is still affected by the resulting estimation errors. Once again, the confidence score reflects the quality of these estimates, with a drop in the confidence score during these events.

While not disclosed in the dataset source, the similarity in term of window number of these events across different runs suggests that they could have been provoked knowingly by the test subject, rather than being natural events. This hypothesis is reinforced by the fact that this dataset was originally intended to evaluate the performance of a HR algorithm under various conditions, potentially including cases where such events could occur. However, without further information, it is impossible to confirm this hypothesis.

Table 2.3 presents the performance metrics of our algorithm on the IEEEPPG dataset [18]. To ensure consistency with the original evaluation, the main metric used to assess the algorithm's performance is the MAE between the estimated HR and the reference HR provided by the dataset, with the mean confidence score used as a secondary metric.

For easier visualization, the MAEs are colored-coded according to their values. ■ Green is used for MAEs below 2 BPM, ■ Orange for MAEs between 2 BPM and 5 BPM, and ■ Red for MAEs above 5 BPM.

Run	MAE [BPM]		Confidence [%]
	Theirs	Ours	
1	1.25	3.58	86
2	1.41	1.93	78
3	0.71	1.28	88
4	0.97	1.62	93
5	0.75	2.25	95
6	0.92	14.51	81
7	0.65	4.40	83
8	0.97	1.66	95
9	0.55	2.50	95
10	2.06	33.36	63
11	1.03	1.95	89
12	0.99	2.51	82
All	1.02	5.96	
Avg N-1	0.93	3.47	
Avg N-2	0.88	2.36	

(a) IEEEPPG training performances.

Run	MAE [BPM]		Confidence [%]
	Theirs	Ours	
1	3.54	15.33	56
2	9.59	8.17	72
3	2.57	3.91	81
4	2.25	4.24	70
5	3.01	3.24	77
6	2.73	1.14	94
All	3.95	6.01	
Avg N-1	2.82	3.13	
Avg N-2	2.64	3.13	

(b) IEEEPPG testing performances.

Table 2.3: Mean absolute error and confidence score for each run on the IEEEPPG training and testing datasets.

Strict comparisons of performance between runs should be treated with caution, as Temko et al. refer at several points in their paper to a dataset of 23 recordings, whereas the IEEEPPG dataset currently available [18] contains only two sets of

12 and 6 recordings, for a total of 18 runs. While they provide a link to the source of the dataset they used, the link is no longer valid and that dataset is no longer available, suggesting that some trimming and splitting may have occurred. Therefore, the performance metrics presented in table 2.3 are based on the available IEEEPPG dataset, which may not be identical to the one used by Temko et al. in their original paper.

Considering the results for the training dataset in table 2.3a, the algorithm performs reasonably well, with an overall MAE of 5.96 BPM across all recordings and an average MAE of 3.47 BPM when excluding the two worst-performing recordings. As expected, the more challenging test dataset yields a higher overall MAE of 6.01 BPM and an average MAE of 3.13 BPM when discarding the two worst-performing recordings, as shown in table 2.3b. For both datasets, the average confidence score is consistent with the expected performance, with lower values for missing or inaccurate estimations and higher values for accurate ones.

While the performance achieved here is lower than that of Temko et al., outperforming their algorithm is not a goal of this thesis. Rather, the aim is to provide a more efficient, portable and configurable implementation of their approach. In its current state, our configuration parameters have not been tuned to this dataset, as MAs produced by FFs do not contain quite the same patterns, frequency ranges and amplitudes as the treadmill running present in the IEEEPPG dataset. A more representative dataset of FFs MAs would be more appropriate for tuning the algorithm, and would likely yield more relevant performance metrics once the parameters are optimized for this specific use case.

2.4 Future improvements

Several improvements could be considered to further enhance the performance, robustness, and usability of the proposed algorithm.

2.4.1 Specific datasets for firefighters

The main obstacle to adapting the algorithm to specific FFs applications is the lack of representative datasets. Even though the algorithm has been validated on synthetic signals and on the IEEEPPG benchmark, these sources do not capture the exact motion and operational context of real firefighting tasks. As a result, the parameter tuning is limited by the absence of ground-truth recordings where PPG, ACC and HR references are available under realistic firefighting conditions.

This limitation is not only a validation issue, but also a design constraint: the

algorithm cannot be fully optimized for FF-specific MAs compensation without observing the actual distributions of their frequency content, amplitude, and temporal patterns. In practice, the most important improvement is therefore to record a dedicated dataset during real FF activities, with synchronized physiological and inertial data, so that the signal-processing pipeline can be tuned to the real operating environment rather than to an external benchmark.

In addition, a task-specific dataset would allow for the development of a more realistic synthetic data generator, which could be used to further validate and optimize the algorithm. This would enable the creation of a more robust and reliable HR estimation system for FFs.

2.4.2 Computational optimizations

Currently, the algorithm recomputes the Hann window coefficients shown in figure 2.2 for each new estimation. This is inefficient, as the coefficients remain constants for a given window size. Precomputing these coefficients and storing them in memory would eliminate the need for repeated calculations, thereby reducing the computational cost during runtime.

In the current implementation of the algorithm, sensor values are provided in a human-readable floating-point format. While this simplifies the development process, it also introduces computational overhead. Switching to integer representation and using an integer DFT instead of a floating-point DFT could reduce the estimation's execution time and improve energy efficiency.

The DFT is a computationally intensive operation, especially for larger window sizes. The `microfft` crate used in the current implementation profides an optimized yet purely software-based implementation of the DFT. Using a hardware-accelerated DFT implementation, if available on the target platform, could further improve the execution speed, increase power efficiency and reduce processor workload.

2.4.3 Algorithmic improvements

Looking beyond the original algorithm, several improvements could be made to the algorithm itself. For instance, the purpose of the linear regression used to estimate the HR is to derive a more reliable HR value from noisy estimates. This is precisely the type of problem addressed by a Kalman filter, which could be used as a post-processing step to provide a more accurate and robust HR estimation. Additionally, using such a filter would allow the confidence score to be directly derived from the covariance matrix of the Kalman filter state, resulting in a more principled and less heuristic-reliant confidence metric.

A known issue with the current algorithm is that the Wiener filter can become too aggressive in reducing the PPG bins affected by MA, potentially removing useful signal components as discussed in section 2.3.2. This problem is difficult to address, because while the PPG and ACC signals are correlated, they have vastly different amplitudes since they represent different physical phenomena. A potential solution could be to use a more sophisticated filtering technique in the Wiener filter step, such as a multi-channel Wiener filter or a more advanced adaptive filtering method.

2.4.4 Algorithm configuration and adaptation

The algorithm's strength partly lies in the fact that it is highly configurable, and could be adapted to different end users. To facilitate the configuration of the algorithm for a specific user within a group, a tool could be developed to automatically determine more suitable parameters from the average parameters for this activity.

Some runs in the IEEEPPG dataset show that the algorithm can become trapped in a local maximum when an error event occurs. A conditional reset mechanism could be implemented to prevent this from happening, by resetting the algorithm when the confidence score remains too low for an extended period, or upon user request. This could enable the algorithm to recover from these situations and continue tracking the HR accurately.

2.4.5 Generator improvements

The generator used to create synthetic data for algorithm validation could be improved to better simulate realistic motion patterns. Currently, the MA is projected onto the three orthogonal ACC axes using fixed coefficients, which may not accurately represent the complex movements of a real subject. Allowing the generator's 3D MA axis to move and rotate over time would create more realistic motion patterns, thereby better simulating the movements of a real subject and potentially revealing additional edge cases to consider.

Along with this improvement to the 3D projection, the modeling of the MA could be enhanced to increase the realism of the generated artifacts. This could involve incorporating more complex motion patterns, simulating different types of activities, or introducing additional noise sources to better reflect real-world conditions. These new artifact sources could be extracted from a FFs-specific dataset and used to improve the validation and performance evaluation of the algorithm.

2.5 Algorithm conclusion

In this chapter, the design and implementation of the HR estimation algorithm has been presented. The algorithm is based on the work of Temko et al., with modifications to improve its performance and adaptability to different scenarios. The algorithm has been implemented in Rust, with a Python interface for ease of use and integration with other tools.

The performance of the algorithm has been evaluated using both synthetic data generated by a custom signal generator and real-world data from the IEEEPPG dataset. The results show that the algorithm is able to accurately estimate the HR under various conditions, with a reasonable MAE and confidence score.

Several potential improvements have been identified, including FFs-specific dataset capture, computational optimizations, algorithmic enhancements, and better configuration and adaptation to specific users or activities. These improvements could further enhance the performance, robustness, and usability of the algorithm.

Overall, the proposed algorithm provides a solid foundation for accurate HR estimation in wearable devices, with potential for further development and optimization for specific use cases.

Chapter 3

Prototype implementation and validation

With the MA mitigation and the estimator implemented, the main limitation appears to be the lack of a task-specific dataset to further validate the algorithm. In addition, the algorithm is designed to be used in a wearable device, which requires a specific hardware platform and software architecture. This chapter presents the prototype implementation of the algorithm on a wearable device, which will first be used to further train and refine the estimator and may later serve as a proof of concept for the final product.

This chapter covers the design choices made during the realization of the prototype, including both hardware and software aspects. It also presents the characterization of the current state of the prototype, and the possible improvements that could be made to enhance its performance and usability.

3.1 Design choices

In order to develop a wearable prototype on which the algorithm could run, a hardware platform first had to be chosen. This platform had to be capable of running the algorithm in real time while also interfacing with the selected sensors.

To perform the necessary data collection and real-time control over this hardware platform, a software architecture had to be designed as well. This architecture had to meet the timing constraints of the use case, while also being able to handle data collection and processing in a modular and maintainable way.

The proof of concept developed in theprevious thesis was based on a MAX32630FTHR microcontroller unit (MCU) rapid prototyping platform and used a MAX30101WING PPG sensor daughterboard for the optical measurements. This sensor breakout

board was selected because it provides a simple-to-use, low-cost and non invasive way to measure HR with 2.54 mm pitch connectors.

As for the software framework, the previous thesis used Mbed OS v5.9.7 with its built-in preemptive priority-based scheduler. Mbed is a free, open-source embedded operating system specifically designed for developing applications on ARM Cortex-M microcontrollers. It provides a simple and user-friendly interface for managing tasks, memory, and peripherals, making it a popular choice for rapid prototyping and embedded systems development. This real-time operating system (RTOS) unfortunately officially reached end-of-life in July 2026, meaning that it no longer receives any support and is therefore strongly discouraged for use in newer software stacks.

From an algorithmic perspective, the previous thesis used a C/C++ port of the HeartPy toolkit [24]. While this algorithm was able to run in real time on the MAX32630FTHR, it was unable to provide accurate results when the device was worn on the wrist in the presence of MAs, simply because acceleration was not measured at all by this peak-based detection method. Nevertheless, it provided good results when the device was stationary.

3.1.1 Microcontroller unit platform

While the MAX32630FTHR was a suitable choice for the previous thesis, it is no longer recommended for new designs [25]. This is likely due to the acquisition of Maxim Integrated by Analog Devices in 2021, which may have contributed to halting the development of new features and optimizations for this MCU. In addition, the MAX32630FTHR is based on a Cortex-M4 core running at 96 MHz, which is insufficient to run our algorithm in real time.

Some of the requirements from the previous thesis are retained, but have their priorities updated.

The first requirement is sufficient memory and processing throughput to run the algorithm in real time, which is a hard timing constraint and the main priority.

Secondly, the platform must be able to interface with the chosen sensors. This requirement remains present but has a lower priority since most modern MCUs have the peripherals to do so.

Thirdly, the platform should not be overly power-hungry. While this is a requirement, it is not as important as the previous two as this can be mitigated through some of the improvements discussed in section 3.4.

Finally, the platform should have strong support and a large community of developers. This requirement is mainly intended to ensure that the development of

the prototype is not hindered by a lack of documentation or examples, and that the platform will remain supported for a long time.

Based on these requirements, three MCUs families were considered: the STM32 family from STMicroelectronics, the nRF family from Nordic Semiconductor, and the ESP32 family from Espressif Systems. Each family offers a broad range of MCUs with different features and capabilities, and all are well supported by their respective manufacturers and communities.

Among the considered families, the STM32 family was selected for this work due to the maturity of the STM32 ecosystem, both in terms of hardware capabilities and software support. The other considered families are not retained because they offered less favorable trade-offs for this specific application. The Nordic nRF family provides excellent power efficiency and integrated wireless connectivity, but its primary design focus is low-power applications rather than maximizing computational throughput. Since real-time execution of the algorithm is the main constraint, their lower computational capabilities made them unsuitable for this application. The ESP32 family provides integrated connectivity features and high processing performance with support for Xtensa and RISC-V instruction sets, but its architecture is primarily aimed at internet of things (IoT) applications, making the software development kit (SDK) heavier both in terms of memory footprint and development complexity.

As processing capability is the primary criterion for this choice, the Nucleo STM32F767ZI in figure 3.1 meets our requirements well [13]. The embedded ST-LINK serial wire debug (SWD) probe makes it easy to flash and debug code. Its onboard MCU is part of the STM32 F7 series, which is designed for high-performance applications. It utilizes an ARM Cortex-M7 core running at up to 216 MHz [26], providing substantially higher computational throughput than the Cortex-M4-based MAX32630FTHR used in the previous thesis. This increase in performance is essential to satisfy the real-time constraints of the proposed algorithm. In addition, the microcontroller provides 512 kB of SRAM and 2 MB of Flash memory, which are sufficient for the implementation of the algorithm and its associated data buffers. Although it also satisfies the hardware interfacing requirements of the prototype by providing multiple inter-integrated circuit (I^2C) interfaces for communication with the selected sensors, it is not optimized for ultra-low-power operation compared with some alternatives. Its power consumption remains acceptable for this prototype, where computational performance is the primary design objective. Furthermore, the STM32 platform benefits from extensive documentation, long-term support from STMicroelectronics, and a large developer community, reducing development effort and supporting the long-term maintainability of the project.

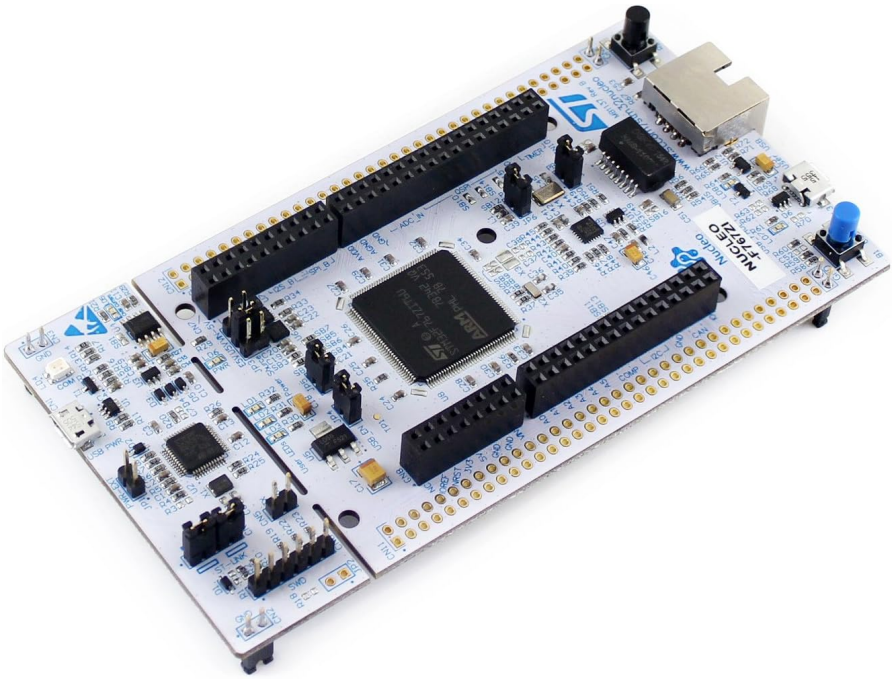


Figure 3.1: Nucleo-144 STM32F767ZI development board.

Although not considered a selection criterion, this platform also provides a wide range of additional features, such as multiple universal asynchronous receiver-transmitter (UART) and serial peripheral interface (SPI) interfaces, a rich set of timers and analog peripherals, and support for various communication protocols. These features may be useful for future expansions of the prototype or for integrating additional functionalities.

At first glance, this Nucleo development board may appear overspecified for the intended application, but this choice is deliberate. At this stage of the project, the algorithm specifications are not yet fully finalized, and its computational and memory requirements may evolve during development. A conservative safety margin is therefore adopted to ensure that future modifications can be accommodated without requiring a redesign of the hardware platform.

3.1.2 Photoplethysmography sensor

As expected, our prototype requires a PPG sensor to measure blood volume pulses. The MAX30101 was the sensor used in the previous thesis, and it is retained for this prototype as it meets the requirements of a minimum sampling frequency of 100 Hz and I^2C communication, along with the availability of extensive documentation and ease of use. In addition, the MAX30101 is a widely used sensor that has been

adopted in many research projects, and it is likely that the IEEEPPG dataset was recorded using this sensor.

While better sensors are available on the market today, the choice of the MAX30101 allows us to compare our results with previous work and attribute the performance of the prototype to the algorithm rather than to the PPG sensor itself.

Despite this, the MAX30101 is still available on the market and can be purchased from various vendors. The MAX30101 breakout board used in this prototype is the MAX30101WING as seen in figure 3.2, which uses 2.54 mm pitch connections and provide a simple-to-use, low-cost and non-invasive way to measure HR. It can be easily interfaced using the I^2C protocol and features a configurable hardware interrupt to either signal the availability of new data or raise an alert.

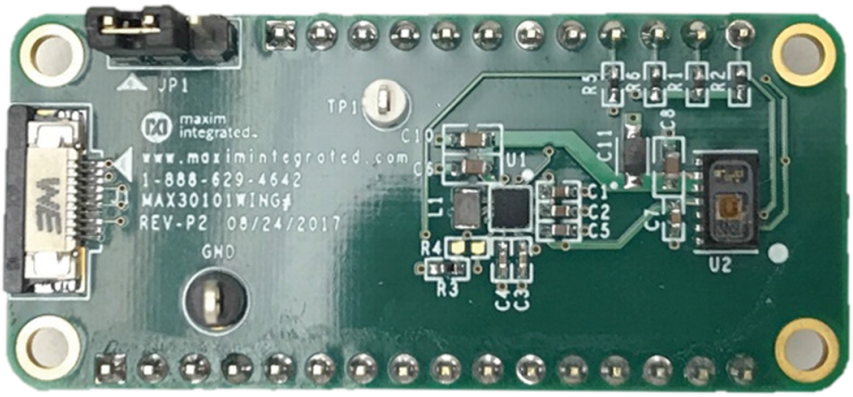


Figure 3.2: MAX30101WING breakout board for the MAX30101 PPG sensor.

3.1.3 Accelerometer sensor

Taking into account the sensor fusion aspect of the algorithm, our prototype also requires an ACC sensor. It is used to measure the motion of the device and is the only source of motion data used later in the MA compensation step.

To be suitable for this work, the ACC sensor must support the I^2C interface, have a sampling frequency of at least 100 Hz and be well documented. Since this prototype is intended to be used as a wearable dataset recorder rather than as a final product, the choice of sensors was also based on the availability of development boards and ease of integration.

The LIS2DW12 was selected because it meets these specifications and benefits from extensive documentation and reference examples provided by STMicroelectronics. In addition, its presence on the X-NUCLEO-IKS01A3 motion MEMS and environmental sensor expansion board, obtained from the university stocks,

greatly simplifies prototyping and integration, as the sensor can be evaluated on a well-established development platform rather than through a custom hardware design [27], [28]. Figure 3.3 shows the X-NUCLEO-IKS01A3 expansion board.

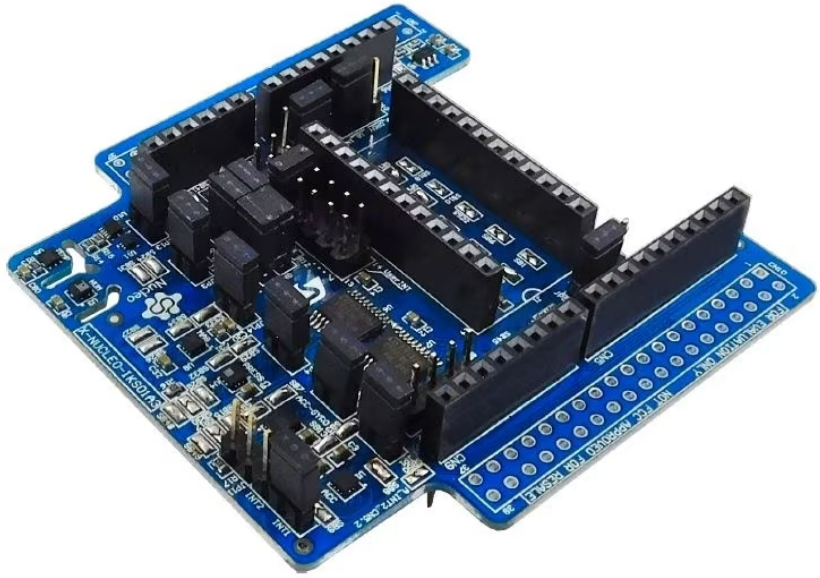


Figure 3.3: X-NUCLEO-IKS01A3 motion MEMS and environmental sensor expansion board.

As of 2026, the X-NUCLEO-IKS01A3 expansion board is listed as "Obsolete" and "Out of production" by STMicroelectronics [28]. When inquirint about alternative Nucleo hats, STMicroelectronics refers to their X-NUCLEO-IKS4A1, a newer expansion board focused on edge artificial intelligence (AI) computing that does not offer enhanced MEMS sensors support. Despite the end-of-life of the X-NUCLEO-IKS01A3, the LIS2DW12 sensor is still available as a standalone component and is marked as "Active" by STMicroelectronics [27], suggesting that the situation is more likely related to a printed circuit board (PCB) redesign and a change in product lineup specialization than to sensor obsolescence.

3.1.4 Software language choice

This prototype also requires an embedded runner, referred to as EmberGuard in this manuscript, as presented in figure 2.6. This runner is responsible for polling and collecting data from the sensors, packaging it in a unified format and feeding it to the estimator. For similar reasons to those expressed in section 2.2.1,

the embedded codebase is written in Rust to leverage its memory safety and performance characteristics.

3.1.5 Embassy asynchronous runtime

Embassy is an asynchronous embedded runtime for Rust that provides a hardware abstraction layer (HAL) and a lightweight executor for running tasks without the overhead of a full operating system [29]. In this prototype, it is particularly useful because the firmware must coordinate several independent activities concurrently: handling the MAX30101 and LIS2DW12 interrupts, polling their data, buffering samples and feeding the estimator in a timely manner. By structuring these operations as cooperative tasks, Embassy keeps the control flow responsive and deterministic while preserving a modular and maintainable codebase.

Compared with a conventional RTOS such as FreeRTOS, Embassy serves the same general purpose of managing concurrency and timing on a microcontroller. Both approaches allow multiple software activities to coexist, provide timing primitives, and separate application logic from low-level hardware interactions. The main difference lies in their execution models: classical RTOSs typically rely on preemptive scheduling with separate task stacks, whereas Embassy uses cooperative asynchronous tasks executed by a single lightweight scheduler. This results in lower overhead, a reduced memory footprint and simpler control of input/output (I/O)-bound operations, which is advantageous for a compact research prototype with a well-defined workload.

In this project, the choice of Embassy is therefore motivated not only by its compatibility with Rust and support for modern embedded development, but also by the fact that the firmware does not require the full complexity of a traditional RTOS. The application mainly requires predictable task coordination, efficient interrupt handling, and straightforward integration with the sensor drivers, all of which can be achieved with a lighter and more explicit programming model. This makes Embassy a practical compromise between raw bare-metal programming and the heavier structure of a conventional RTOS.

3.1.6 Modern logging with defmt

Traditional logging methods such as `printf` are not well suited to embedded systems due to their more higher ressources usage. These methods tytpically store the raw format strings in flash memory, process them with the corresponding arguments at runtime and and then send the resulting messages through a serial interface. This is not only slow, but also requires significant memory for the format

strings and output buffer. To address this, the codebase uses the `defmt` crate, which provides modern and efficient logging capabilities for embedded systems. With this crate, the format strings are not stored as plain text in the firmware image. Instead, they are replaced by compact identifiers, while the information required to reconstruct and format the messages is retained by the receiving system. Consequently, the full format strings do not need to be stored in or uploaded to the embedded device, significantly reducing its memory footprint.

This logging is used to transfer data to the computer for further analysis, and among other things allow internal variables such as the PPG and ACC samples, the estimated HR and display the PXX/PNN/PSS spectra.

3.1.7 Toolchain and build support

To facilitate the development process and avoid environment inconsistencies, this prototype uses a devcontainer setup that provides a reproducible development environment. The container is based on the official Microsoft Rust devcontainer image and is extended with the host-side tools required for embedded development, including USB utilities, build dependencies, the ARM cross-compiler, and the `probe-rs` flashing tool. This setup also installs the corresponding Cortex-M Rust target and grants the development user access to USB devices, ensuring that the same toolchain, flashing workflow and permissions are available on any machine supporting Docker-based containerization.

3.2 Implementation

3.2.1 Nucleo clock setup

Before putting the Nucleo board into operation, the clock configuration must be set up to ensure that the microcontroller operates at the desired frequency. This frequency should be high enough to allow the estimator to run in real time and to meet the hard timing constraints associated with reading the sensors first in first out (FIFO) buffers before their samples are overwritten.

Figure 3.4 shows how the clock is configured. The Nucleo STM32F767ZI board receives an 8 MHz clock signal from the onboard ST-LINK debugger. This signal is used as the external `HSE` high-speed clock source and is first divided by 4 before being supplied to the phase-locked loop (PLL). The resulting 2 MHz signal is multiplied by 216 within the PLL, producing a 432 MHz voltage-controlled oscillator frequency. This frequency is then divided by 2 to produce a 216 MHz `SYSCLK`, which is used as the system clock for the MCU core. The `SYSCLK` is subsequently

divided by 1 to obtain the 216 MHz `HCLK`. This `HCLK` clocks the `AHB` bus, while the `APB1` and `APB2` peripheral buses are derived from it using division factors of 4 and 2, respectively, resulting in clock frequencies of 54 MHz and 108 MHz. In addition, the PLL provides a 48 MHz output through its Q divider and a 108 MHz output through its R divider, which can be used as clock sources for peripherals requiring dedicated clock inputs.

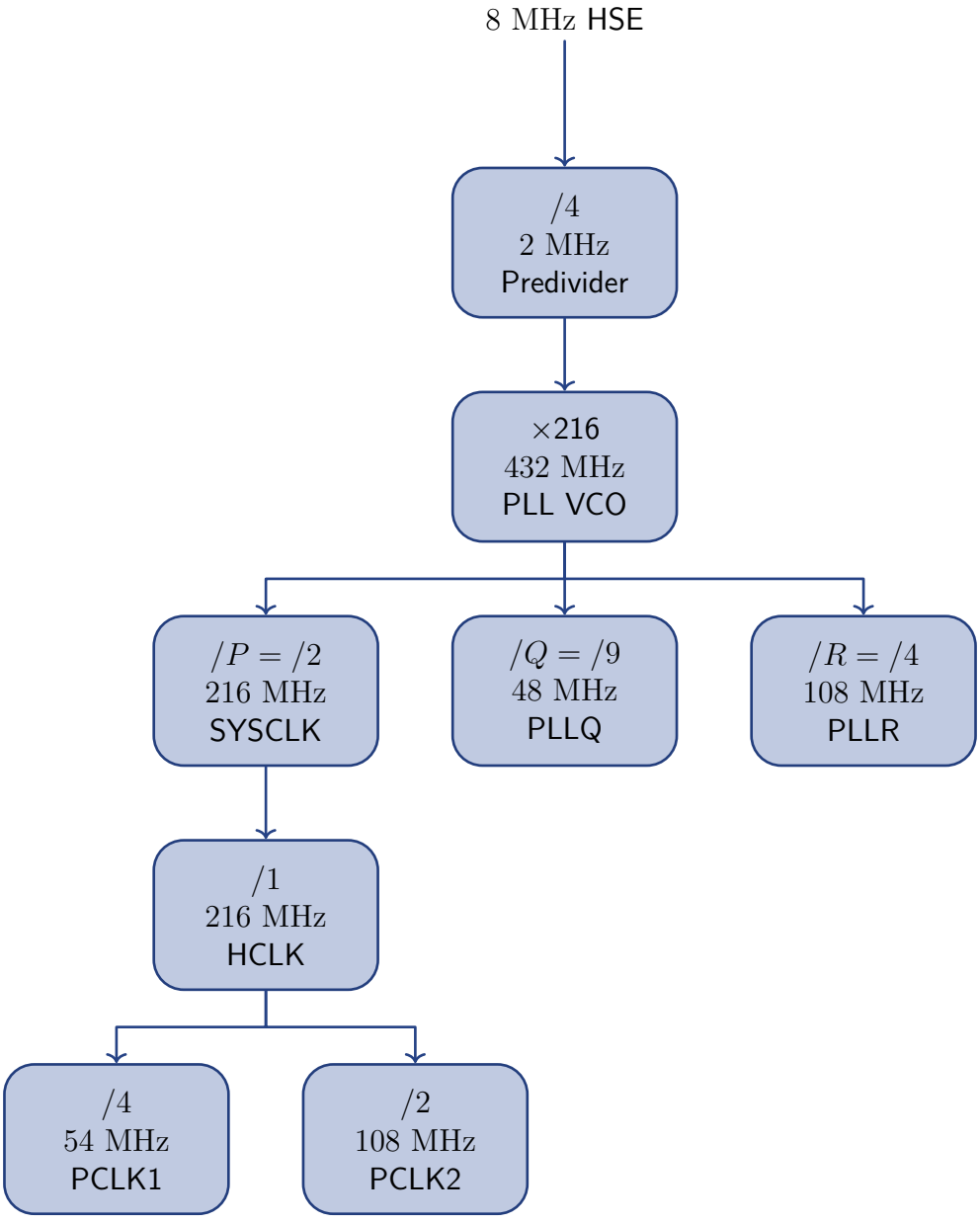


Figure 3.4: Nucleo clock configuration diagram.

The choice of using the maximum allowed frequency of 216 MHz for the HCLK is motivated by the need to maximize the processing throughput of the MCU core. Once the HR tracking algorithm reaches a finalized state, the processing requirements may be reduced, allowing for a lower clock frequency to be used. This would result in lower power consumption and lower heat generation, which are important considerations for a wearable device. However, at this stage of development, the priority is to ensure that the algorithm can run in real-time without any performance bottlenecks.

3.2.2 Prototype assembly and wiring

With the MCU platform and the two sensors selected, the prototype can now be assembled. Since both the host and the LIS2DW12 sensor development board are from STMicroelectronics, they allow for quick and straightforward integration, as the expansion board can be directly mounted on the Nucleo board using the Zio header pair. For the MAX30101 breakout board, a simple breadboard and jumper wires are used to connect it to the same Zio header pair via the X-NUCLEO-IKS01A3 expansion board thanks to the 2.54 mm pitch of its headers.

Following the X-NUCLEO-IKS01A3 expansion board datasheet [28], the jumpers are configured to enable the LIS2DW12 sensor, connect its INT line to the PF3 pin via the Zio connector, and link its I^2C interface to the common I^2C bus, and consequently to the Nucleo board. These I^2C lines are routed to the PB8 and PB9 pins of the Nucleo host, which are configured as the I^2C1 line. While the bus connecting the Zio connector to the LIS2DW12 sensor is referred to as I^2C2 in the expansion board datasheet, this work uses the Nucleo board's naming convention for consistency with the codebase running on it.

Other sensors present on the expansion board may or may not be powered, but this does not affect any key metric of the prototype as the overall power consumption is not assessed neither optimized in this work.

Regarding its configuration, the MAX30101WING datasheet [30] specifies that a 3.3 V supply voltage must be provided to the 3V3 pin, the GND pin must be connected to the common ground and pins 1 and 2 must be bridged from JP1 to set the I/O lines to 3.3 V logic level. The SCL and SDA lines are then connected to PB8 and PB9 pins the Nucleo board using jumper wires, corresponding to the I^2C1 interface of the Nucleo board. The EN pin is connected to the 3V3 supply to permanently enable the sensor, while the INT pin is connected to the PA6 pin of the Nucleo board to receive hardware interrupts from the sensor.

Figure 3.5 shows the assembled prototype. The individual components can be

identified from figure 3.1, figure 3.3 and figure 3.2.

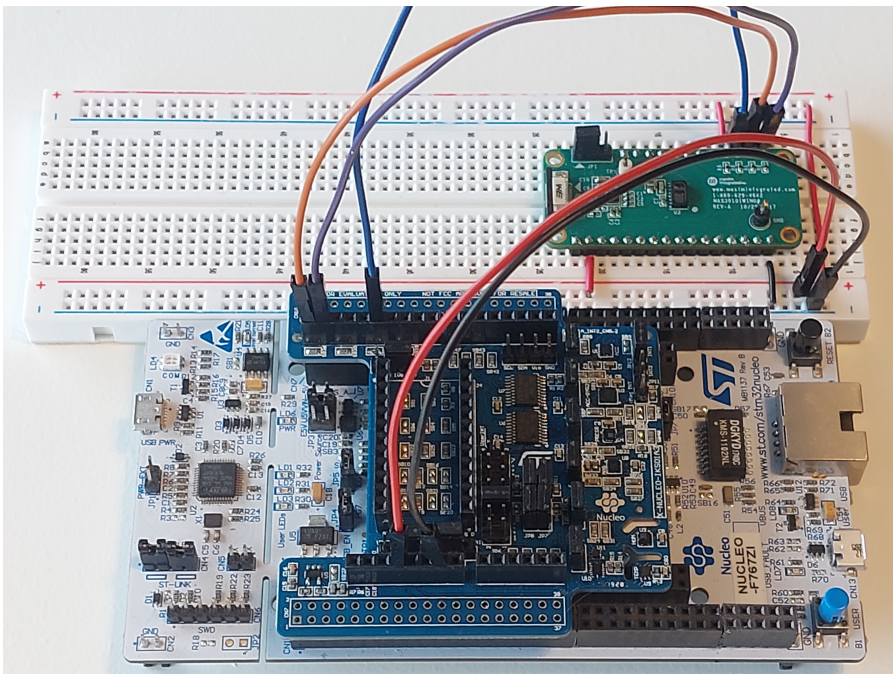
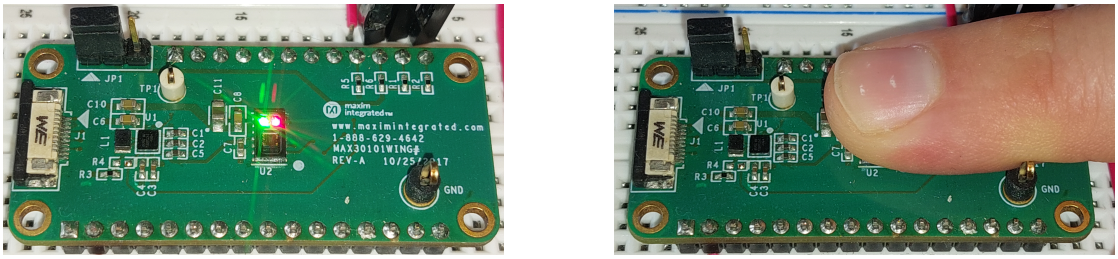


Figure 3.5: Prototype assembly.

Figure 3.6 shows the MAX30101WING breakout board in the same configuration as that used in the current setup. In figure 3.6a, the green and red LEDs can be seen illuminated. Figure 3.6b shows the same board with a finger placed on top of the combined LEDs and photodiode assembly, allowing the PPG sensor to perform measurements.



(a) PPG sensor without a finger. (b) PPG sensor with a finger on it.

Figure 3.6: MAX30101WING PPG in use.

Despite having ordered the MAX30101WING after the Analog Devices acquisition of Maxim Integrated, the received unit bears a Maxim Integrated logo, a

revision indicator of “REV-A” and a date code of “10/25/2017”. These indicators suggest that the board was manufactured before the official launch of the MAX30101WING in March 2018 [30], likely making it a pre-production unit. This is not an issue per se, as the datasheet and reference design are still valid, but it explains why the sensor and some test loops are not located in the same positions as in figure 3.2.

Figure 3.7 shows a block diagram of the prototype shown in figure 3.5, summarizing the electrical connections between its components. Each component is represented using the PCBs solder mask color. The bus and net colors are chosen to match those of the corresponding jumper wires in figure 3.5, where applicable.

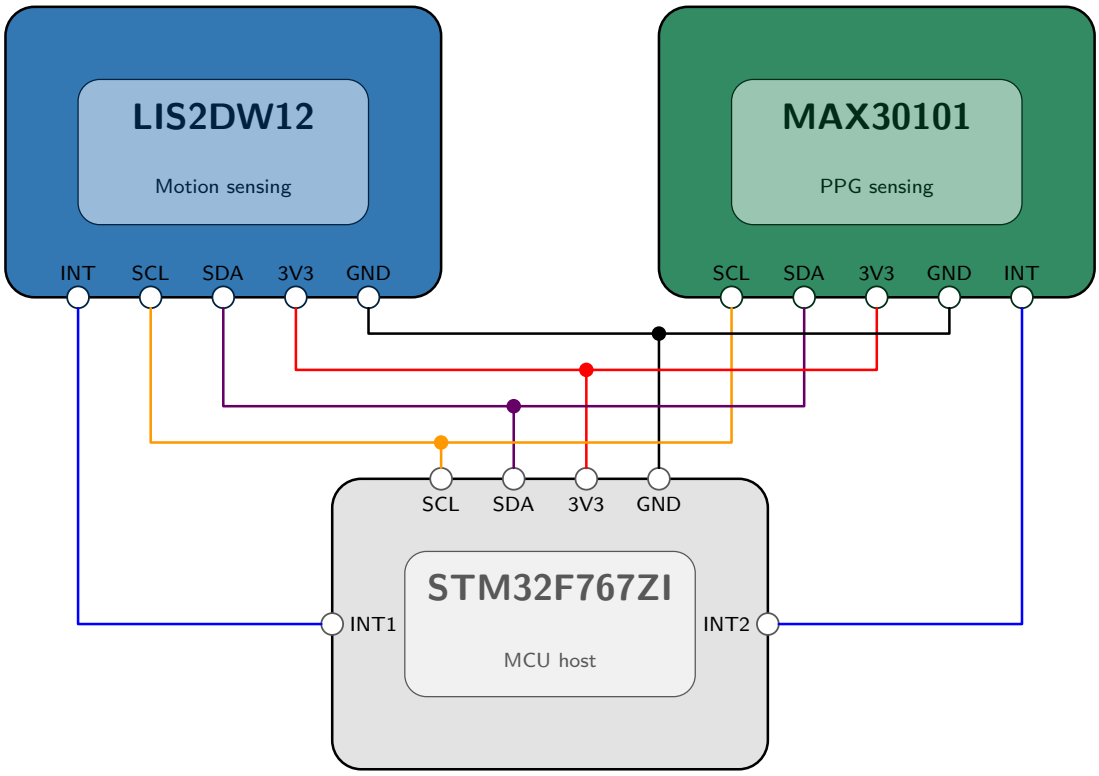


Figure 3.7: Prototype PCB blocks and their exposed pins.

With this representation, it is easy to see that the I^2C bus allows the two sensors to share the same communication lines, connecting them in parallel for data transfer and power intake. This bus sharing is possible because the I^2C protocol allows multiple devices to communicate on the same bus using their respective addresses. The `INT` lines are the only elements breaking out of this parallel arrangement, as they are connected to different pins on the Nucleo board to allow for independent

interrupt detection and handling. The MAX30101 `INT` line is open-drain and active-low, so it is pulled up to the 3.3 V supply voltage. The LIS2DW12 `INT` line is push-pull and active-high, so it is pulled down to the common ground to ensure a defined logic level when the sensor is not asserting the interrupt.

3.2.3 Sensor interfacing

The first step in EmberGuard's task loop is to collect PPG and ACC samples from the sensors. To do so, the embedded codebase relies on two Rust crates that provide a high-level interface to the underlying hardware while abstracting away the low-level details of I^2C communication and register configuration. These crates are designed to work with the Embassy asynchronous runtime and are therefore compatible with the cooperative, non-blocking task model used in the firmware. The ACC driver is availables on crates.io as `lis2dw12-pid-rs` [31], while the PPG driver is available as `max30101-rs` [32].

Together, these drivers provide a common `no_std` abstraction layer that is largely independent of the specific MCU or HAL. Their design relies on Rust's trait system to standardize the driver interfaces, allowing the same high-level logic to be reused across different hardware backends. This approach makes the embedded software more portable and interoperable than a typical C-based setup, where drivers portability often depends more heavily on the underlying platform and available software ecosystem. The crates are also modular and reusable: they support both blocking and asynchronous operation, and they can be used with either shared or dedicated communication buses. Their hardware abstraction is built on the `embedded-hal` [33] and `embedded-hal-async` [34] interfaces, which define common operations for I^2C and SPI communication. A shared register-transport layer then provides a uniform register-based API across supported protocols, while optional shared-bus support allows multiple peripherals to access the same physical bus without conflicts. Although SPI communication is not used in this thesis, the LIS2DW12 driver also supports SPI, further highlighting the portability of the driver abstraction.

LIS2DW12 interfacing

For the ACC sensor, STMicroelectronics provide a C driver distributed under the BSD-3-Clause license [35]. This driver provides a blocking interface to the sensor and is not compatible with the Embassy asynchronous runtime. To address this, the driver has been ported to Rust and extended with an asynchronous interface in the `lis2dw12-pid-rs` crate [31]. The roadmap for porting this driver to Rust was to first create the `error.rs`, `reg.rs` and `transport.rs` files to respectively define the

possible error types, host the register addresses and provide an abstraction between raw values and named enums and define the transport abstraction interface. Using these abstractions, the `lis2dw12.rs` file was then created to implement the logic required to translate high-level commands into the low-level register operations.

The driver architecture was designed and implemented using placeholders, with one implementation fully provided for each type of functionality. The rest of the driver was then built around this structure and filled in with the actual logic using GitHub Copilot within the Visual Studio Code IDE.

To ensure that the driver behaves as expected, a parity test suite was developed to validate its functionality. It leverages the Rust's ability to call C code through `extern "C"` bindings, invoking the original LIS2DW12 C driver and comparing the resulting register state and returned values with those of the Rust port.

MAX30101 interfacing

For the PPG sensor, no official driver has been published by either Maxim Integrated or Analog Devices. Consequently, this meant that a driver had to be developed from scratch based on the information provided in the datasheet. The resulting implementation is published as the `max30101-rs` crate on crates.io [32]. The development process first involved defining the abstractions in the `error.rs`, `reg.rs` and `transport.rs` files, followed by implementing the logic in the `max30101.rs` file to translate high-level commands into low-level register operations. This last file was created using the same approach as for the LIS2DW12 driver, by defining the structure and implementing one feature as an example, then using GitHub Copilot to implement the remaining functionality. The same feature set is also provided to support either blocking or asynchronous operation, as well as shared or dedicated communication buses. Unlike the LIS2DW12, the MAX30101 only supports I^2C communication, so no SPI support is provided by this driver.

Since no official driver is available for comparison, no parity test suite could be developed. This made it more difficult to ensure that the driver behaves as expected, and required a more thorough review of the datasheet and the implementation to verify the driver correctness. The driver was tested by running the prototype and verifying that the sensor was able to provide valid data and that the data was consistent with the expected behavior of the sensor. However, this does not guarantee that the driver is fully correct and complete, as not all of the sensor's features are used in the current implementation.

3.2.4 Sensor setup

Both sensors have a set of configuration registers that can be programmed to adjust their behavior. During the initialization phase of the prototype, these registers are

configured to set the desired sampling frequency, resolution, and other parameters.

In this context, a distinction is made between the raw sensor sampling rate and the effective sampling frequency. The former is defined by the rate at which the sensor physically acquires samples, while the latter refers to the post-averaging sample stream written to the FIFO and consumed by the processing pipeline.

The effective sampling frequency of both sensors is set to 100 Hz, which is an integer multiple of the 25 Hz downsampling frequency used by the estimator before converting the data to the spectral domain. This allows for an accurate representation of the signal, while also reducing the amount of data that needs to be processed by the estimator.

Instead of having to retrieve one sample every millisecond, the sensors contain a FIFO buffer that can hold up to 32 multidimensional samples that can be read in a single burst I^2C transaction. A suitable threshold can be set to trigger an interrupt when the FIFO buffer reaches a certain level, known as the "water level". This allows the prototype to read multiple samples at once, reducing the number of I^2C transactions and improving the overall efficiency of the data collection process.

The water level is set to 25 samples for both sensors, meaning that a hardware interrupt should be triggered every 250 ms, while still leaving a margin of 70 ms before the FIFO buffer is full and begins to overwrite older samples. This margin represents a hard timing constraint, as an overwriting of the FIFO buffer would result in a shift in the estimator's input data, leading to artifacts in the PXX or PNN spectra, and consequently in the HR estimate.

The MAX30101 is a multi-wavelength PPG sensor that can measure the blood volume pulse using different light sources. In this implementation, the red and green light sources with wavelengths of 660 nm and 527 nm respectively are used because they both provide information about the blood volume pulse, With the red light also providing information about the blood oxygen saturation. Blood oxygen saturation is not studied in this work, but the red light is still used to provide sensing diversity in the PPG signal.

3.2.5 Prototype peripheral usage

In addition to the sensors and core processing, the prototype utilizes onboard peripherals to provide a visual status indication of its operating status. The Nucleo STM32F767ZI board includes three user-controllable LEDs.

The green LED acts as a heartbeat indicator and is toggled once per second, providing a continuous indication that the firmware is running and responsive. This serves as a simple sanity check to ensure the system has not entered a deadlock or

became unresponsive.

The blue LED is activated during every HR estimation cycle and remains on for the duration of the estimation computation. This visual feedback allows an observer to monitor the timing of the estimation process and verify that the system is performing estimations at the expected frequency.

The red LED has been used during the development phase to indicate error conditions, such as sensor communication failures or unexpected states in the firmware. It is no longer in use in the latest version of the codebase.

3.2.6 Task design and scheduling

The complex task of running the estimator on the collected data is broken down into smaller tasks that are executed cooperatively. Each task is responsible for a specific aspect of the data collection and processing pipeline, and they communicate with each other through shared data structures and synchronization primitives provided by the Embassy runtime. Figure 3.8 shows the data collection and processing tasks pipeline.

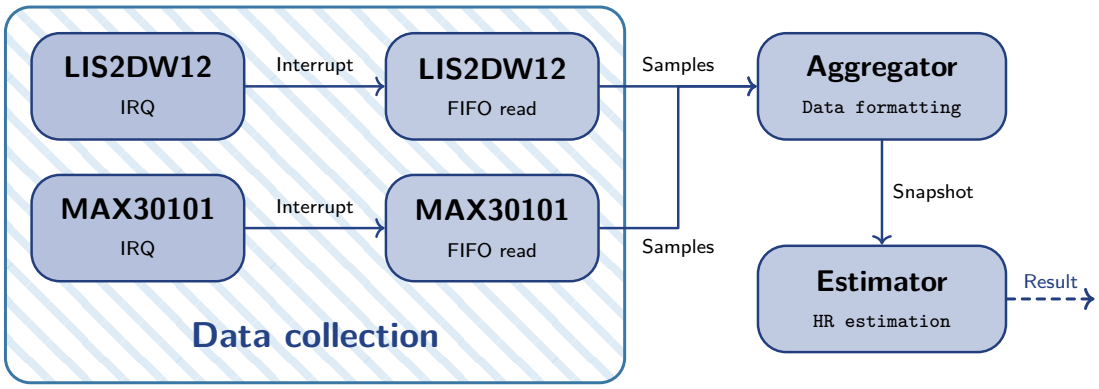


Figure 3.8: Data collection and processing tasks pipeline.

First, the sensor interrupts trigger sensor-specific interrupt request (IRQ) handlers, which unlock corresponding data retrieval tasks. These tasks read the samples present in the internal FIFO buffers and pass them to the shared formatting task. This task is solely responsible for formatting the raw data into the 8 s record expected by the HR tracking algorithm. Once this task has enough samples to append a 2 s window to the previously used buffer, it copies its circular buffer to the estimator's input buffer and then rotates it so that the oldest samples are positioned at the beginning of the buffer. The estimator task is then unlocked and can process the new data, producing an updated HR estimate.

This task-based architecture allows for a clear separation of concerns, making the codebase more maintainable and easier to understand. It also facilitates the addition of new tasks in the future, such as user reporting, wireless data transmission, or security and self-checking features.

Figure 3.9 shows a theoretical waterfall diagram of the tasks execution and scheduling. To make the figure clearer, a time cut is represented by dashed horizontal lines between the first data collection and the last ACC IRQ before the estimator is unlocked.

This figure represents the execution for the first round of data collection after 250 ms, and again once the buffer has been primed with 8 s of data. After this initial priming time, an estimation is performed every 2 s.

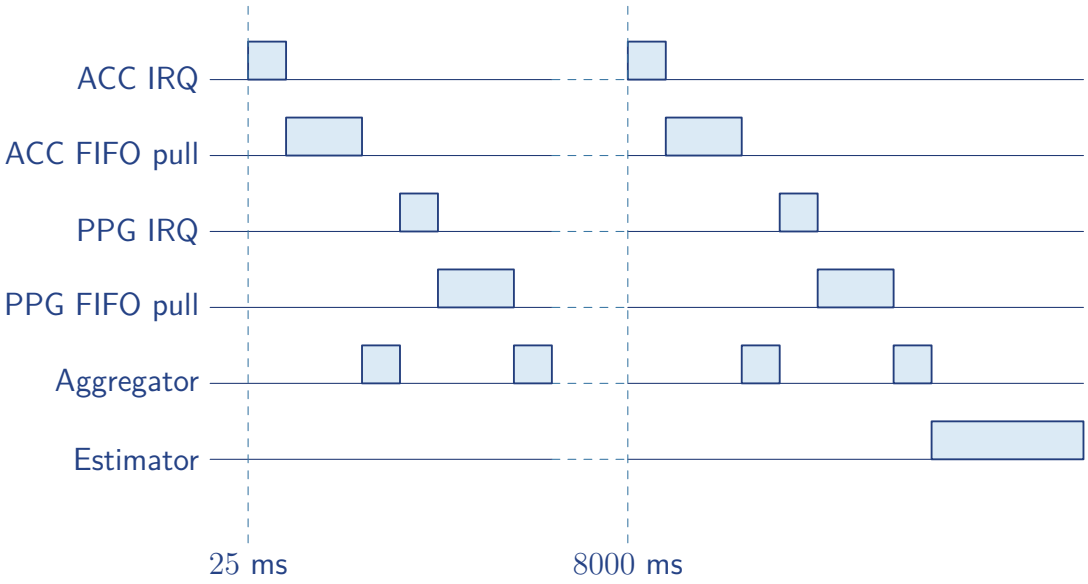


Figure 3.9: Simulated waterfall diagram of the embedded codebase.

The sensors trigger an interrupt when their internal FIFO reaches the configured threshold, which unlocks the corresponding data retrieval task. Once a sensor value has been retrieved, the formatting task is unlocked to process the new sample and prepare it for the estimator. When enough new data has been collected, the estimator task is unlocked to compute an HR estimate.

The interrupt handlers have a short execution time, as they only unlock another task. The data retrieval tasks have a longer overall execution time, but the majority of this time is spent waiting for the I^2C bus to transfer the data from the sensors. Since `embassy-stm32` performs I^2C bus operations asynchronously using direct

memory access (DMA) through its `embedded-hal-async` implementation, tasks can yield control while transactions are ongoing, allowing other tasks to run in the meantime. These two factors, along with many instances where the tasks willingly yield control to the scheduler, allow for a smooth and efficient task execution without bottlenecks or task starvation.

3.3 Characterization

Having implemented the prototype, both in terms of hardware and software, it is now possible to characterize its performance metrics. This characterization is essential to validate the design choices made during the development process and to ensure that the prototype meets the requirements of the HR tracking algorithm.

This section covers three main aspects: the HR estimation accuracy achieved by the prototype, the real-time task execution and scheduling performance, and the system resource utilization in terms of central processing unit (CPU) workload and memory.

3.3.1 Heart rate estimation performance

Moving from the simulation to the prototype, it is important to validate that the HR tracking algorithm still performs as expected. This validation ensures that both the algorithmic pipeline and the sensor interfacing are working correctly, and that the prototype can provide reliable HR estimates in real time.

In its current form factor, the prototype is not suitable for tests involving movement, as the PPG and ACC sensors are not mechanically coupled. This means that the ACC sensor does not measure the movement responsible for the MAs in the PPG signal, and therefore cannot be used to compensate for them.

As the MAX30101WING breakout board is designed for finger measurement, the board layout does not optimize the placement of the configuration jumper and the test point loops for wrist measurements. This arrangement makes it impossible to achieve good skin contact when the board is placed under the wrist, as these elements prevent the PPG sensor from making direct contact with the skin.

These limitations mean that the prototype can only be tested in a static configuration, where the PPG sensor is placed on a finger as shown in figure 3.6b.

For all the following tests, the reference HR is obtained from a smartwatch worn on the subject's right wrist, as medical-grade equipment is not available for these tests. This consumer-grade device provides an approximate HR estimate using its own PPG sensor and algorithm. Although it may be less accurate than

medical-grade equipment, it serves as a reasonable reference for the purpose of this test.

To retrieve the PSS spectra back on the computer for display and analysis, the embedded codebase uses the `print-samples` feature to send the sensor samples to the computer via the `defmt` logging crate. The data is then captured by the host computer and converted into a .ts format, which is used as input for the simulator detailed in section 2.2.2. Since the simulator runs the same estimator as the embedded codebase and has access to the `std` Rust ecosystem, it allows for the PNN, PXX and PSS spectra to be computed and displayed, along with the HR estimates.

Because the same embedded system is responsible for both the real-time `defmt` logging and the estimator pipeline, the added data extraction overhead is not negligible. Each burst of raw sensor samples must first be serialized and forwarded to the host computer before the corresponding spectra can be reconstructed offline, which consumes processing time and bus bandwidth that would otherwise be available for the acquisition and formatting tasks. As a result, the estimator is not producing HR estimates at the ideal cadence of one update every 2 s. This explains why the measured number of HR estimates is lower than the theoretical value, and why the experimental curves are still plotted against window number rather than absolute time in seconds.

Figure 3.10 shows the PSS spectrum and the corresponding HR estimates obtained from a steady-state test, without the legend as in figure 2.10 and figure 2.11. In this test, the test subject is seated and instructed to place their right index finger on the PPG sensor, while remaining still and relaxed for a total of 1 minute.

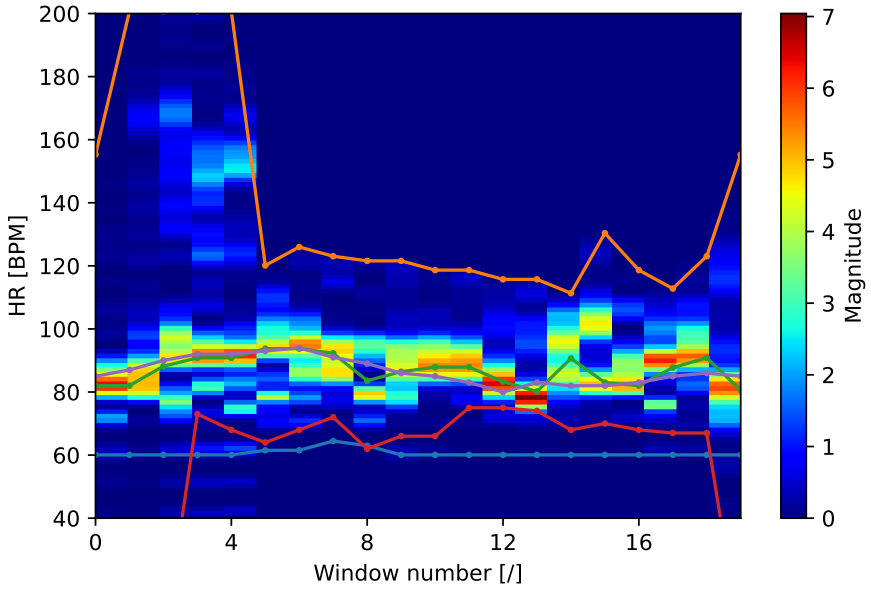


Figure 3.10: PSS spectrum and HR estimates in a steady state.

As expected, the HR estimation trends remain stable and consistent with the subject's actual HR, with a MAE of 3.28 BPM. An offset between the estimated and reference HR is observed, which can be explained by the fact that the newest sensor data is largely attenuated because of the Hann window applied to the PPG signal, resulting in a theoretical delay of 4 s between the reference and estimated HR. Notably, an offset closer to 2 s between the estimated and reference HRs is measured in this test, indicating that the smartwatch reference may itself have an offset of approximately 2 s. This is not unexpected given the consumer-grade nature of the device, especially when compared with the medical-grade equipment used to record the reference signal in the IEEEPPG dataset. After manually compensating for this offset, the MAE is reduced to 2.75 BPM, which is still higher than the results presented in table 2.3a, but remains within the expected range for this non-optimized prototype sensor setup.

The confidence score is significantly lower than expected, particularly for the dissipation component. This could be explained by the fact that the ACC sensor is not measuring any movement, and that scaling its spectrum by its standard deviation results in a noisy spectrum that may cause overcompensation during the MA cancellation step, as discussed in section 2.3.1.

Figure 3.11 shows the PSS spectrum and the corresponding HR estimates obtained during the second test, with only the estimated HR and reference values

shown for clarity. During this test, the subject is comfortably seated with their right index finger placed on the PPG sensor. The subject is first asked to relax for one minute, after which they perform a series of exercises with their left hand using weights for another minute to increase cardiac activity and HR. This cycle of resting and exercising is repeated for a total duration of 5 minutes. Once again, the legend is omitted for clarity, but the colors correspond to those in the legend of figure 3.10.

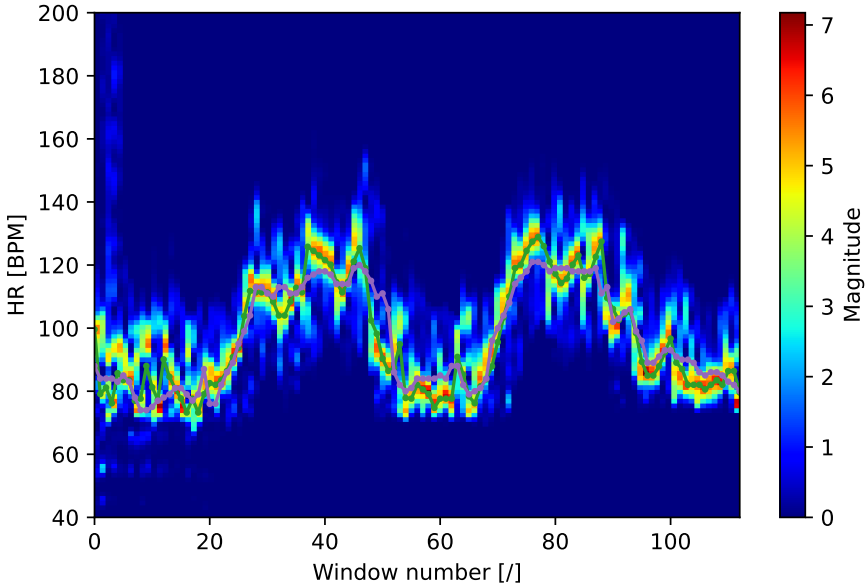


Figure 3.11: PSS spectrum and HR estimates under exercise.

Even with the induced physical activity, the HR estimates remain stable and consistent with the reference values, with a MAE of 3.92 BPM after manually compensating for the offset. This result is valuable, as it demonstrates that the prototype can still provide reliable HR estimates while the subject is performing physical activity, despite the subject's high stress level, which leads to a higher instantaneous HR change rate, as well as the non-ideal sensor placement and configuration.

Despite this performance, more noise is observed in the spectrum than in the previous test. This is likely due to the fact that during movement, the subject's finger may exert varying pressure on the PPG sensor, directly affecting the PPG common-mode signal and thus making the MA cancellation less effective. This is a limitation of the current prototype, and it is expected to be absent in a wearable form factor where the PPG sensor is strapped to the subject's wrist via an elastic

band, ensuring more consistent contact with the skin and thus a more stable PPG signal.

3.3.2 Task execution timing

Figure 3.12 shows a waterfall diagram equivalent to figure 3.9 after the initial priming period, with task execution times measured on the running prototype. Since the aggregator task is unlocked and executed briefly after each sample retrieval, it is omitted from the figure for clarity.

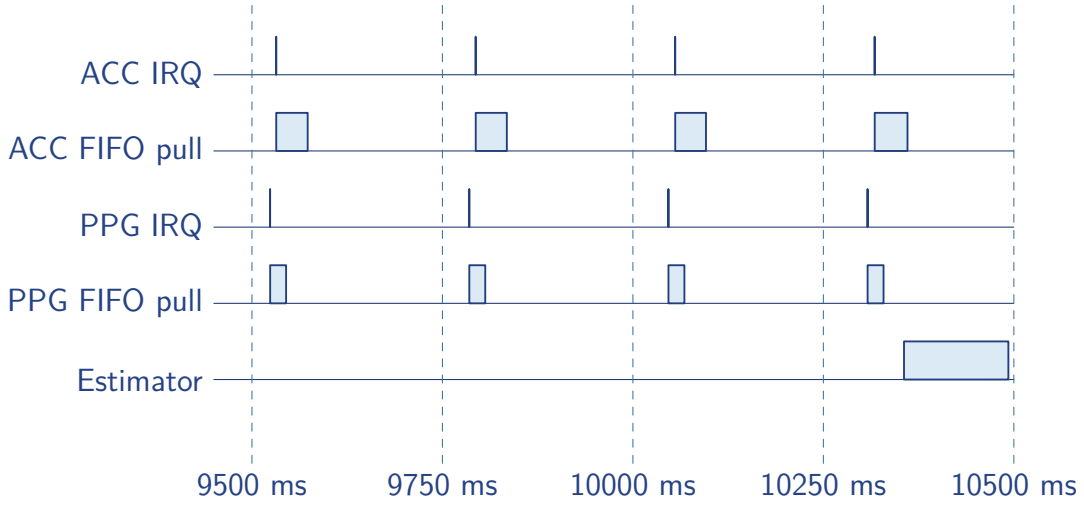


Figure 3.12: Measured waterfall diagram from task traces.

Besides the aggregator task, the execution order of the tasks is comparable to that observed in the simulation, with the IRQ handlers executing first, followed by the FIFO data retrieval task of the corresponding sensor.

On average, once the estimator task is unlocked, it takes nearly 140 ms to perform an estimation, which is significantly shorter than the 2 s period between two consecutive estimations. It is, however, longer than the 70 ms time margin between a sensor FIFO buffer reaching its water level and beginning to discard older samples. Because of this, the estimator task implements both a blocking and an asynchronous implementations with multiple yield points, allowing it to be interrupted by other tasks during execution. This ensures that the estimator task does not block the data retrieval tasks, which could lead to data loss and thus a fracture in the continuity of the estimator's input data.

Although the ACC data retrieval task appears to have a longer execution time than the PPG data retrieval task, their actual execution times are comparable.

This apparent difference is due to how the execution time is measured: the start time is recorded when the task is unlocked, while the end time is recorded after the last sample has been passed to the aggregator. When the ACC IRQ handler unlocks the task, the I^2C bus is still locked by the PPG data retrieval task. Consequently, the ACC task spends part of its measured execution interval waiting for the I^2C bus to become available before beginning its own data transfer. This waiting time makes the ACC task appear longer in the waterfall diagram, despite its actual data-transfer execution time being comparable to that of the PPG task.

Looking more closely at the period of the IRQ handlers, it can be observed that they do not fire exactly every 250 ms, as what would be expected with a FIFO water level of 25 samples and a sampling frequency of 100 Hz. Instead, the IRQ handlers fire approximately every 265 ms. This discrepancy can be explained by the fact that the sensors use their own internal clock to sample the data, which may deviate from the Nucleo board's clock. In our case, the measured interval between consecutive interrupts is 15 ms longer than expected, which corresponds to a timing error of 6%. This deviation can be attributed to the combined frequency offsets of the sensors and MCU clocks.

This same clock offset is also observed in the first test, and partly explains the observed 113 estimations in 5 min instead of the expected 147 estimations.

3.3.3 Resource usage

Using a task trace of the running prototype, the execution times of all tasks are measured and summarized in table 3.1.

Metric	Value
Total execution time	27.4 s
Busy time	6.44 s
Idle time	20.7 s
CPU usage	23.8%

Table 3.1: Prototype CPU usage.

The total execution time is the span between the start of the first measured task and the end of the last measured task, while the busy time is the sum of the execution times of all tasks, removing the overlaps the tasks wait for each other rather than executing concurrently. The idle time is the difference between the total execution time and the busy time, and represents the time spent waiting for events such as sensor interrupts or peripheral availability.

This CPU usage is relatively low, indicating that the prototype has sufficient processing power to handle the current workload. While this metric is relevant for overall evaluation, it is not the most critical factor in this context. The primary concern is ensuring that the system can meet its hard real-time constraints, which are essential for avoiding data loss.

Using the `arm-none-eabi-size` command-line tool, the memory usage of the prototype is measured and summarized in table 3.2. The different sections of the compiled program are grouped into two main categories: flash memory and random access memory (RAM).

Metric	Size	STM32F767ZI	Usage
Flash	141 KB	2048 KB	6.9%
RAM	144 KB	512 KB	28.1%

Table 3.2: Prototype memory resource usage.

Flash memory usage primarily consists of the program code and read-only data, as well as the flash-resident portions associated with initialized data. RAM usage consists primarily of initialized and uninitialized global and static data. The reported RAM usage does not directly account for stack and heap consumption at runtime, which depends on the application's execution and memory configuration. A more detailed analysis of stack and heap usage can be performed using specialized tools, but this is beyond the scope of this work.

The flash memory usage is relatively low, leaving ample space for either pre-computing and storing additional constants or switching to a less expensive MCU with less flash memory.

As for RAM usage, it is higher than expected, but still leaves enough space for additional tasks to be added in the future. The current RAM usage is primarily due to the large arrays and buffers used in the estimator, which are essential for storing the input data and intermediate results.

3.4 Future improvements

Despite achieving a good performance with the current prototype, there are still several areas for improvement that can be explored in future works. These improvements can enhance the performance, usability, and robustness of the system, making it more suitable for real-world applications.

3.4.1 Enhance stability and robustness

While putting memory safety at its core, Rust does not provide any guarantees regarding the absence of deadlocks or unexpected behavior. At many instances in the code, the `unwrap()` method is used to retrieve the value contained in an `Option` or `Result` type, without specifying how to handle the case where the value is an `Err` error variant. This can lead to a panic and a subsequent system reset, which is not acceptable in a life-critical application. To enhance the stability and robustness of the system, it is important to avoid using `unwrap()` calls and instead handle errors gracefully, either by returning an error to the caller or by implementing a recovery mechanism.

Another approach to enhance the stability and robustness of the system would be to add a watchdog task that periodically wakes up and performs a sanity check on the system. If the watchdog task detects that the system is in a deadlock or an unexpected state, it can trigger a reset of the system to recover from the error. This would provide an additional layer of protection against unexpected behavior and ensure that the system remains operational even in the presence of errors.

In this context, the safety of the system is more critical than any marginal optimization, as a failure in the embedded execution loop could compromise the reliability of the complete measurement pipeline.

3.4.2 Evolve into a wearable form factor

The current prototype suffers in many respects from its non-wearable form factor. The PPG sensor is not in direct contact with the skin, and the ACC sensor does not measure the movement responsible for the MAs in the PPG signal. This makes it impossible to test the HR tracking algorithm under realistic conditions, as the sensors do not measure the same signals as they would in a wearable device.

In other words, a wearable implementation would not only improve usability, but would also strengthen the validity of the experimental results by reproducing the conditions in which the algorithm is expected to operate in practice.

3.4.3 Use hardware acceleration

For now, the estimator does not use many hardware acceleration features. Using more dedicated hardware peripherals could free up the MCU to perform other tasks and reduce overall CPU usage and power consumption.

Once the sensing setup is made more representative and the software is made more robust, this becomes a natural next step toward a more efficient embedded implementation, especially if the prototype is later ported to a lower-power platform.

3.4.4 Grant wireless monitoring capabilities

Finally, the current prototype does not have any wireless communication capabilities, which limits its usability in real-world applications. Adding a wireless communication module, such as a ZigBee mesh link or a LoRaWAN module would allow the prototype to transmit HR estimates to a central monitoring system, thereby getting better use of the HR tracking algorithm. This would enable remote monitoring of the subject's HR, and bring the project closer to a real-world application.

This type of enhancement is attractive from a deployment perspective, but it remains secondary to the more fundamental issues of system reliability and realistic physiological measurement conditions.

3.4.5 Floating point representation

The current implementation uses the `f32` floating-point representation for the data. Migrating to a fixed-point representation could reduce the RAM usage and allow for a lower clock frequency, thereby reducing power consumption. However, this would come at the cost of a reduced dynamic range and precision, which may affect the accuracy of the HR estimates. A careful analysis of the trade-offs between precision, dynamic range, and resource usage would be necessary to determine the optimal representation for the data.

Although this optimization may be valuable for a production-oriented design, it is less urgent than the improvements above, since the current prototype already has sufficient memory and processing headroom for the demonstrated workload.

3.5 Prototype conclusion

In this chapter, the design and implementation of the prototype have been presented, along with its characterization and potential future improvements. The prototype serves as a proof of concept for the HR tracking algorithm, demonstrating its feasibility and performance in a real-world setting.

The characterization of the prototype has shown that it can provide reliable HR estimates in both steady-state and exercise conditions, with a reasonable MAE compared to the reference values. The task execution timing and resource usage analysis have also provided insights into the performance and efficiency of the system, highlighting areas for potential optimization.

Overall, the prototype represents a significant step towards the development of a wearable device capable of accurately tracking HR in real time, and lays the groundwork for future improvements and enhancements to further refine its performance and usability.

Chapter 4

Conclusion

This thesis developed and validated a motion-artifact-robust HR monitoring system for embedded deployment in firefighting contexts. Building on the previous work of Martin Brans, it addresses the challenge of maintaining accurate physiological monitoring despite severe motion disturbances encountered during firefighting operations.

Contributions

The primary contribution is a frequency-domain HR tracking algorithm that mitigates MAs while satisfying real-time embedded constraints. Key enhancements include Hann windowing to reduce spectral leakage, configurable search-range processing with a minimum 5 BPM radius to track sudden changes in HR, and a dual-component confidence score for reliability assessment. The algorithm was implemented as a modular Rust crate (HeartTrack), making it suitable for reuse across different deployments.

Critically, this algorithm improves upon Martin Brans' previous work by incorporating ACC-based MA compensation. The previous approach was unable to track HR accurately during motion because it did not measure acceleration data. This thesis directly addresses that limitation through dual-sensor fusion.

This algorithm is validated through a two-stage approach: first through simulation using synthetic and real-world data from the IEEEPPG dataset, and second through deployment on the EmberGuard embedded system running on an STM32F767ZI microcontroller responsible for PPG and ACC data aggregation and real-time HR estimations. The system successfully demonstrates reliable real-time estimation performance, with a MAE of 2.75 BPM in steady-state conditions and 3.92 BPM during physical activity, with the prototype stationary and the test

subject performing controlled exercise. Estimation cycles complete respecting the timing constraints, and performance remains stable during exercise scenarios. Resource utilization remains well below microcontroller's capacity, with approximately 141 KB of flash and 144 KB of RAM used on a deviwe with 2 MB of flash and 512 KB of RAM.

Practical impact

This work provides a critical foundation for FF safety systems. It demonstrates the feasibility of continuous, motion-robust physiological monitoring through real-time HR tracking, a capability that is essential for future systems enabling early detection of overexertion and objective health assessment. The complete implementation has been developed under permissive licenses and is fully documented. This includes `HeartTrack` and `EmberGuard`, which will be made available publicly as development progresses, as well as the custom sensor drivers already published for the MAX30101 [32] and LIS2DW12 [31]. This establishes a cost-effective and reproducible approach suitable for deployment and future iteration across firefighting units.

Limitations

While the algorithm has been validated against the IEEEPPG dataset, a general-purpose physiological monitoring benchmark, it has not been tuned to FF-specific motion patterns. The algorithm parameters were not optimized from fire service datasets, as no representative dataset containing synchronized PPG, ACC, and HR ground-truth recordings is available for actual firefighting operations. Consequently, field validation under real firefighting conditions is essential before deployment.

The reflectance PPG proved to deliver accurate measurements during the laboratory tests, but performance variations across different skin tones and sensor placements have not been explored. Prototype validation was limited to setups in whitch the EmberGuard platform stayed static while the subject performed controlled exercise. The hardware was deliberately over-specified to provide development flexibility, but final production systems would benefit from optimization for lower-power platforms. Multi-day monitoring scenarios and performance under extreme environmental conditions such as thermal stress and smoke-induced optical scattering remain untested.

Future work

The highest priority is the collection of FF-specific datasets to enable algorithm optimization for fire service operations. Systematic field trials with firefighting teams would validate algorithm performance under representative task conditions, capture authentic motion patterns, and enable parameter tuning for fireground-specific MAs. Concurrently, the PPG signal could be extended to estimate blood oxygen saturation, respiratory rate, and potentially blood pressure using the available sensor data. Real-time anomaly detection could be implemented to identify potentially dangerous physiological deviations. Platform optimization could enable deployment on lower-power hardware while maintaining real-time constraints. Wireless communication integration, using technologies such as ZigBee or LoRaWAN would be required for final product deployment, enabling remote monitoring of personnel health status by command centers. Finally, a hardware redesign enabling wearable integration, with secure sensor mounting on the wrist or torso would be necessary for practical field deployment.

Summary

This thesis demonstrates that accurate physiological monitoring in high-motion environments is achievable through systematic attention to signal processing and real-time embedded constraints. The motion-artifact-robust HR monitoring system developed for firefighters moves beyond a proof of concept to a functional prototype validated on general-purpose activity data. Although the algorithm was not optimized for firefighter-specific data, its general design suggests potential applicability to military operations, rescue services, extreme sports, and clinical rehabilitation, although each application would require domain-specific validation. By enabling continuous, ACC-compensated HR monitoring, the system provides a foundation for safer decision-making during high-risk interventions. With the addition of FF-specific dataset collection and field validation, this foundation could support the development of robust occupational health monitoring systems capable of contributing to the reduction of injury rates among emergency responders.

References

[1] RAND Corporation. "Emergency responder injuries and fatalities." (2005), [Online]. Available: https://www.rand.org/content/dam/rand/pubs/technical_reports/2005/RAND_TR100.pdf (visited on 04/06/2026).

[2] National Institute of Standards and Technology. "The economics of firefighter injuries in the united states." (2019), [Online]. Available: https://doi.org/10.6028/NIST.TN.2078 (visited on 04/06/2026).

[3] European Forest Fire Information System (EFFIS). "Effis statistics portal: Estimates (european union, 2026 vs 2006–2025)," Copernicus Emergency Management Service, Joint Research Centre. (2026), [Online]. Available: https://forest-fire.emergency.copernicus.eu/apps/effis.statistics/estimates (visited on 08/14/2026).

[4] M. Brans, "Enhancing the safety of firefighters in intervention with a wearable heart rate sensor," EPL - UCLouvain, 2025.

[5] Microsoft. "Visual studio code," Microsoft. (), [Online]. Available: https://code.visualstudio.com/ (visited on 08/16/2026).

[6] Git Project. "Git," Git Project. (), [Online]. Available: https://git-scm.com/ (visited on 08/16/2026).

[7] GitHub, Inc. "Github," GitHub, Inc. (), [Online]. Available: https://github.com/ (visited on 08/16/2026).

[8] Atlassian. "Gitflow workflow," Atlassian. (), [Online]. Available: https://www.atlassian.com/git/tutorials/comparing-workflows/gitflow-workflow (visited on 08/16/2026).

[9] The Rust Project Developers. "Rust programming language," The Rust Project Developers. (), [Online]. Available: https://www.rust-lang.org/ (visited on 08/16/2026).

[10] ISO C++ Foundation / cplusplus.com. "C++," ISO C++ Foundation. (), [Online]. Available: https://isocpp.org/ (visited on 08/16/2026).

[11] Python Software Foundation. “Python,” Python Software Foundation. (), [Online]. Available: https://www.python.org/ (visited on 08/16/2026).

[12] Docker, Inc. “Docker,” Docker, Inc. (), [Online]. Available: https://www.docker.com/ (visited on 08/16/2026).

[13] STMicroelectronics. “Stm32 nucleo-144 development board with stm32f767zi mcu,” STMicroelectronics. (), [Online]. Available: https://www.st.com/en/evaluation-tools/nucleo-f767zi.html (visited on 08/14/2026).

[14] Wikipedia contributors. “Photoplethysmogram — wikipedia, the free encyclopedia.” (), [Online]. Available: https://en.wikipedia.org/wiki/Photoplethysmogram (visited on 05/14/2026).

[15] T. Tamura, Y. Maeda, M. Sekine, and M. Yoshida, “Wearable photoplethysmographic sensors—past and present,” *Electronics*, vol. 3, no. 2, pp. 282–302, 2014. DOI: 10.3390/electronics3020282. [Online]. Available: https://www.mdpi.com/2079-9292/3/2/282.

[16] M. Lee, Q. Wei, S. Lee, and H. Park, “Ddm-hsa: Dual deterministic model-based heart sound analysis for daily life monitoring,” *Sensors*, vol. 23, Feb. 2023. DOI: 10.3390/s23052423.

[17] Wikipedia contributors. “Qrs complex — wikipedia, the free encyclopedia.” (2026), [Online]. Available: https://en.wikipedia.org/wiki/QRS_complex (visited on 05/21/2026).

[18] C. W. Tan, C. Bergmeir, F. Petitjean, and G. I. Webb, *Ieeeppg dataset*, 2020. DOI: 10.5281/zenodo.3902710. [Online]. Available: https://doi.org/10.5281/zenodo.3902710.

[19] A. Temko, “Accurate heart rate monitoring during physical exercises using ppg,” *IEEE Transactions on Biomedical Engineering*, vol. 64, no. 9, pp. 2016–2024, 2017. DOI: 10.1109/TBME.2017.2676243.

[20] Wikipedia contributors. “Heart rate — wikipedia, the free encyclopedia.” (), [Online]. Available: https://en.wikipedia.org/wiki/Heart_rate (visited on 05/15/2026).

[21] Wikipedia contributors. “Raised-cosine window,” Wikipedia, The Free Encyclopedia. (2025), [Online]. Available: https://en.wikipedia.org/wiki/Raised-cosine_filter (visited on 07/28/2026).

[22] B. Stroustrup. “Quotes.” (Dec. 6, 2025), [Online]. Available: https://www.stroustrup.com/quotes.html (visited on 07/28/2026).

[23] The Rust Project Developers. “The rust programming language.” (), [Online]. Available: https://doc.rust-lang.org/stable/book/index.html (visited on 08/14/2026).

[24] HeartPy contributors. “Heartpy: Python heart rate analysis toolkit,” Read the Docs. (), [Online]. Available: https://python-heart-rate-analysis-toolkit.readthedocs.io/en/latest/ (visited on 08/04/2026).

[25] Analog Devices, Inc. “Max32630,” Analog Devices, Inc. (), [Online]. Available: https://www.analog.com/en/products/max32630.html (visited on 08/04/2026).

[26] STMicroelectronics. “Stm32f767zi,” STMicroelectronics. (), [Online]. Available: https://www.st.com/en/microcontrollers-microprocessors/stm32f767zi.html (visited on 08/04/2026).

[27] STMicroelectronics. “Lis2dw12,” STMicroelectronics. (), [Online]. Available: https://www.st.com/en/mems-and-sensors/lis2dw12.html (visited on 08/04/2026).

[28] STMicroelectronics. “X-nucleo-iks01a3,” STMicroelectronics. (), [Online]. Available: https://www.st.com/en/evaluation-tools/x-nucleo-iks01a3.html (visited on 08/04/2026).

[29] Embassy contributors. “Embassy: Async/await runtime for embedded systems,” Embassy Project. (), [Online]. Available: https://embassy.dev/ (visited on 08/14/2026).

[30] Analog Devices, Inc. “Max30101wing evaluation board,” Analog Devices, Inc. (), [Online]. Available: https://www.analog.com/en/resources/evaluation-hardware-and-software/evaluation-boards-kits/max30101wing.html (visited on 08/08/2026).

[31] Jérôme Hendrick. “Lis2dw12 rust driver,” crates.io. (), [Online]. Available: https://crates.io/crates/lis2dw12-pid-rs (visited on 08/14/2026).

[32] Jérôme Hendrick. “Max30101 rust driver,” crates.io. (), [Online]. Available: https://crates.io/crates/max30101-rs (visited on 08/14/2026).

[33] embedded-hal contributors. “Embedded-hal,” crates.io. (), [Online]. Available: https://crates.io/crates/embedded-hal (visited on 08/14/2026).

[34] embedded-hal-async contributors. “Embedded-hal-async,” crates.io. (), [Online]. Available: https://crates.io/crates/embedded-hal-async (visited on 08/14/2026).

[35] STMicroelectronics. “Lis2dw12 pid,” STMicroelectronics. (), [Online]. Available: https://github.com/STMicroelectronics/lis2dw12-pid (visited on 08/10/2026).

Appendix A

Use of artificial intelligence

This appendix describes how artificial intelligence (AI) tools were used during the development of this thesis.

A.1 Overview of artificial intelligence tool usage

AI tools were used during the thesis for specific tasks related to software development, technical problem solving, data and log analysis, and the preparation and refinement of the written manuscript. AI-generated or AI-assisted outputs were reviewed and verified by the author before being used.

A.2 Tools employed

TwoAI tools were primaarily used during this thesis:

- **GitHub Copilot:** Integrated into Visual Studio Code, GitHub Copilot was used for code-related tasks, including code generation, code review, debugging assistance, and analysis of software artifacts.
- **ChatGPT:** Used for technical questions and conceptual clarification, as well as for writing support, documentation refinement, and discussions of theoretical concepts. ChatGPT also provided support for exploring ideas and improving the clarity of written content beyond code-specific contexts.

A.3 Specific applications of AI tools

A.3.1 Code development and review

GitHub Copilot was used to assist with the following tasks:

- **Code generation:** Completing routine code tasks following predefined templates and patterns (e.g., function implementations, boilerplate code for embedded systems).
- **Code review and analysis:** Reviewing source code for potential issues and improvements in readability and implementation. Suggestions were reviewed and verified by the author before being incorporated.
- **Log and trace analysis:** Assisting with the parsing and exploitation of log files and task traces to identify relevant patterns and potential anomalies.

A.3.2 Writing and communication

ChatGPT was used for the following tasks:

- **Text refinement:** Suggesting improvements to grammar, clarity, fluency, and academic English, as well as alternative formulations of technical statements, while preserving the intended technical meaning.
- **Organization and structure:** Suggesting possible structures and logical organization for sections and chapters.

A.3.3 Technical problem solving

AI tools were also used for the following technical tasks:

- **LaTeX compilation and formatting:** Assisting with the identification and resolution of compilation errors, formatting issues, and syntax problems.
- **Debugging and documentation:** Assisting with the analysis of error messages and suggesting possible debugging approaches and documentation improvements. Proposed solutions were tested before being adopted.

A.4 Boundaries and limitations

The scientific and engineering decisions underlying the thesis remained the responsibility of the author. This includes the research methodology, system design, algorithm development, experimental validation, interpretation of results, and conclusions. AI tools were used as supporting tools, but their suggestions did not replace the author's technical judgement.

A.5 Verification of AI-assisted outputs

AI-assisted outputs were reviewed and verified as appropriate before being incorporated into the thesis or the software implementation. AI-assisted code was tested and reviewed before being incorporated into the implementation. Textual and technical suggestions were reviewed for consistency with the intended meaning and, where applicable, verified through independent testing or reference to the relevant documentation.

A.6 Transparency statement

The use of AI tools in this thesis is disclosed to provide a transparent account of the extent and manner in which they contributed to the work. The author remains responsible for the final content, code, analysis, and conclusions presented in this thesis.

UNIVERSITÉ CATHOLIQUE DE LOUVAIN
École polytechnique de Louvain
Rue Archimède, 1 bte L6.11.01, 1348 Louvain-la-Neuve, Belgique | **www.uclouvain.be/epl**