sensing,
interaction &
perception lab

Department of Computer Science, ETH Zürich

in collaboration with
Department of Computer Science, University of Salzburg

Bachelor Thesis

VitalVision: A Mobile Platform for Wearable ECG/PPG Validation

Emanuel Mairoll

supervisor

Prof. Dr. Christian Holz
Sensing, Interaction & Perception Lab
ETH Zürich

supervisor

Manuel Meier
Sensing, Interaction & Perception Lab
ETH Zürich

supervisor

Assoc. Prof. Dr. Andreas Naderlinger
Department of Computer Science
University of Salzburg

August 14th, 2024

Emanuel Mairoll
VitalVison: A Mobile Platform for
Wearable ECG/PPG Validation
Bachelor Thesis, August 14th, 2024
Supervisors: Prof. Dr. Christian Holz, Manuel Meier
and Assoc. Prof. Dr. Andreas Naderlinger

ETH Zürich
Department of Computer Science, ETH Zürich
Universitätstrasse 6
8092 Zürich, Switzerland

University of Salzburg
Department of Computer Science, University of Salzburg
Jakob-Haringer-Straße 2
5020 Salzburg, Austria

VitalVison: A Mobile Platform for Wearable ECG/PPG Validation

Emanuel Mairoll
Sensing, Interaction & Perception Lab, Department of Computer Science, ETH Zürich
Department of Computer Science, University of Salzburg
emairoll@student.ethz.ch, emanuel.mairoll@stud.plus.ac.at

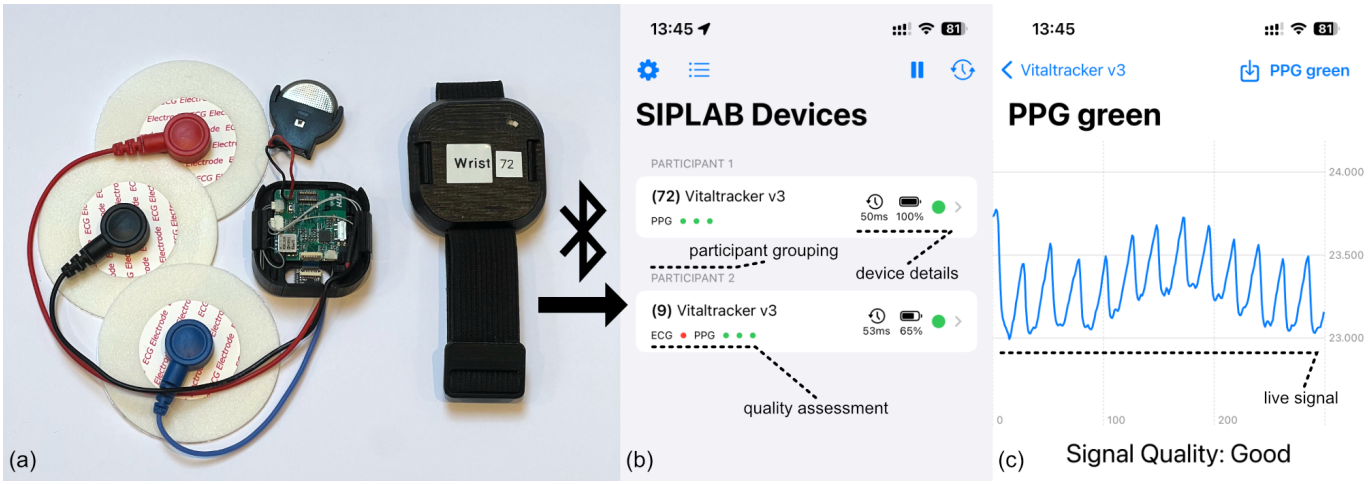


Figure 1: Overview of the VitalVision system for real-time physiological signal validation. (a) Wearable devices equipped with ECG and PPG sensors (SIPLAB VitalTracker). (b) Main screen of the VitalVision app showing an overview of connected devices, their status, quality assessment indicators, buttons for pausing connections, resyncing device time, sorting options, and settings. (c) Live plot of streamed ECG signal with real-time quality assessment.

ABSTRACT

Cardiovascular diseases (CVDs) are the leading cause of death globally, prompting extensive research into continuous physiological monitoring. Wearable devices show promise in detecting and managing CVDs, but further progress requires studies with reliable data recorded in real-world conditions. Maintaining quality during extended in-the-wild studies is challenging due to electrode displacement and motion artifacts. To address this, we introduce VitalVision, a mobile platform for real-time validation of ECG and photoplethysmogram (PPG) data streamed from wearables. Our system includes lightweight algorithms for signal quality assessment, implemented in a Rust-based core library for cross-platform processing and device communication. An iOS frontend provides multi-device monitoring, visualization and alerts. We evaluated our algorithms using recordings of 13 hours of daily activity. The ECG algorithm achieved perfect binary classification accuracy on our dataset, while the PPG algorithm demonstrated 83.33% overall accuracy across three categories. By offloading validation to a smartphone, VitalVision enables continuous quality monitoring during extended studies without sacrificing battery life, allowing longer, more reliable real-world data collection.

Keywords

Wearable devices; Signal quality assessment; Real-time validation; ECG; PPG; Mobile computing; In-the-wild studies; Rust; iOS

INTRODUCTION

The global health landscape is significantly impacted by cardiovascular diseases (CVDs), which remain a leading cause of death worldwide [1]. In response to this significant health challenge, researchers have intensified their focus on developing advanced physiological monitoring techniques. Continuous monitoring of Electrocardiogram (ECG) and Photoplethysmogram (PPG) signals has emerged as a crucial tool in various health studies and applications [2, 3, 4]. These sensors play a vital role in clinical research, enabling researchers to monitor heart conditions [5], detect anomalies [6], and assess overall cardiovascular health [7, 2].

The acquisition of accurate ECG and PPG data is challenging, especially during physical activities or other dynamic conditions. These scenarios introduce disturbances like motion artifacts, sweating and sensor displacement, that can severely

compromise data quality [8]. These challenges manifest differently across various data collection contexts:

1. In laboratory settings, devices can be connected to power sources, allowing for continuous, high-quality data collection and posing no limitation to accurate, real-time quality monitoring. However, these controlled environments may not capture the full range of real-world conditions.

2. For short-term studies in the wild, battery-powered devices can provide some level of feedback and data quality control. However, these studies, often lasting only a few hours, fail to capture representative long-term data crucial for many health studies.

3. Long-term studies in the wild, aiming for 24 to 36-hour continuous monitoring periods, face significant challenges due to battery constraints. Traditional devices struggle to maintain both continuous data collection and real-time feedback over extended periods, often resulting in "blind" data acquisition where quality issues are only discovered after the study concludes.

This creates a significant challenge for researchers aiming to collect high-quality, long-term physiological data in real-world settings. The ideal solution would provide continuous monitoring and real-time feedback without compromising the device's ability to collect data for extended periods.

To address these challenges, we developed VitalVision, a mobile platform for physiological data monitoring. This system leverages smartphone processing power to provide real-time validation and visualization of ECG and PPG data streamed from wearable devices, enabling long-term studies without compromising data quality or device battery life.

Our mobile application connects to multiple wearable devices, streams and plots ECG and PPG data, and uses lightweight, mobile-compatible algorithms to detect quality issues in real time, warning the experimenter if persistent disruptions occur. Importantly, our solution keeps the critical path of data acquisition on the wearable device itself, with the smartphone serving as a display and notification system. This ensures that data recording continues even if the connection to the phone is temporarily lost.

By leveraging the processing power of a more capable device for data validation and visualization, VitalVision allows the wearable devices to focus on their primary task of data acquisition, thus extending their battery life and enabling long-term, uninterrupted data collection.

Contributions

The work in this thesis makes the following contributions.

- We present a mobile-compatible cross-platform framework for connecting and streaming data from multiple wearable devices, optimized for long-term, low-power operation in real-world environments.

- We establish lightweight algorithms for real-time detection and validation of disruptions in ECG and PPG data, designed to run on smartphone hardware.

- We present a user-friendly data visualization frontend for the framework, designed to inform and notify experimenters of data quality issues as they occur.

The complete source code for VitalVision is publicly available at https://github.com/EmanuelMairoll/VitalVision. This repository includes the core library, iOS frontend, build scripts for cross-platform compilation, and evaluation scripts used in this thesis. The project is released under the MIT license, allowing for broad use, modification, and distribution.

RELATED WORK

The assessment of signal quality for ECG and PPG data has been extensively studied, with approaches ranging from simple rule-based algorithms to complex machine learning and deep learning techniques. This section provides an overview of existing methods, focusing on those that inspired our work, and highlights the need for computationally efficient algorithms suitable for real-time mobile processing.

ECG Signal Quality Assessment

ECG waveforms have been clinically studied for decades, leading to the development of numerous Signal Quality Indices (SQIs) such as signal-to-noise ratio, skewness, and kurtosis. Comprehensive reviews by Raman et al. [9] and Daluwatte et al. [10] provide an excellent overview of these metrics and evaluate their effectiveness. These SQIs are often implemented in rule-based algorithms, which rely on extensive signal processing. Reference implementations are often provided, however typically in MATLAB or Python, making them challenging to port to mobile platforms.

Several algorithms have been developed for ECG signal analysis, with the well-known Pan-Tompkins algorithm [11] from 1985 far predating modern wearables. Recent adaptations of this approach, such as those by Neri et al. [12, 13], have provided inspiration for our ECG algorithm.

A large class of machine learning algorithms has also emerged for ECG signal quality assessment. These range from classical machine learning approaches [14] to deep neural networks [15] and convolutional neural networks [16]. While these methods often achieve high accuracy, they are typically too computationally expensive for real-time processing on mobile devices.

PPG Signal Quality Assessment

PPG signal quality assessment faces similar challenges to ECG, but with increased complexity due to greater variability in waveform morphology. Moscato et. al. [17] provides a comprehensive overview of PPG signal quality assessment techniques; the field is dominated by template matching and machine learning approaches.

Template matching methods either use predefined templates [18] or assess self-similarity within the signal [19, 20]. Machine learning approaches for PPG quality assessment include Support Vector Machines (SVMs) applied to extracted SQIs [21, 22], which have shown strong performance. Including accelerometer data into the SVM approach further boosts performance [14]. Deep learning approaches have also been

explored [14], but like their ECG counterparts, these are often too computationally intensive for mobile applications.

For scenarios requiring "low computational efficiency" and less training data, techniques such as the Elliptic Envelope method have been proposed [23].

Summary

Existing methods for ECG and PPG signal quality assessment often prioritize accuracy over computational efficiency, making them less suitable for real-time processing on mobile devices. Our work addresses this gap by developing simple, threshold-based algorithms for both ECG and PPG data.

For ECG, we draw inspiration from QRS detection algorithms like Pan-Tompkins [11], simplifying the method to focus on R-R interval consistency. Recent adaptations by Neri et al. [12, 13] for wearable devices further influenced our approach, though we employ a different peak extraction method.

Our PPG analysis is influenced by Sukor et al. [20], who demonstrated the effectiveness of analyzing pulse morphology for quality assessment. We adapt these techniques, focusing on key morphological features of individual pulses for efficient computation.

Overall, our approach focuses on continuous data assessment for long-term monitoring and therefore prioritizes computational efficiency and ease of implementation on mobile devices, rather than maximizing accuracy.

SYSTEM DESIGN AND IMPLEMENTATION

VitalVision was designed to meet the following core requirements:

- **Real-Time Data Streaming and Visualization**: Immediate feedback through real-time streaming and visualization of ECG and PPG signals.
- **Quality Assessment**: Automatic evaluation of ECG and PPG signal quality, with energy-efficient algorithms suitable for mobile devices.
- **Low-Power Wireless Communication**: Utilization of Bluetooth Low Energy (BLE) for power-efficient data transmission between wearables and the mobile device.
- **Multi-Device Support**: Simultaneous connection to multiple wearables for multi-participant or multi-body-location studies.
- **Channel System**: Flexible support for multiple data streams per device (e.g., ECG, PPG-Red, PPG-Green), enabling future expansion.
- **Configurable Alerts**: Customizable notifications for persistent signal quality issues.
- **Cross-Platform Core**: Platform-independent functionality for potential expansion to various operating systems.
- **User-Friendly Interface**: Intuitive device management, data visualization, and system configuration.
- **Extensible Architecture**: Modular design facilitating integration of new devices, signal types, algorithms, and visualization methods.
- **On-Device Data Processing and Storage**: Ensuring that sensitive health data remains on the mobile device and is processed locally, without leaving the device unless explicitly initiated by the user.

To meet these requirements, VitalVision consists of three main components:

- **Core Library** (Rust): Handles BLE communication, data management, signal analysis, and clock synchronization.
- **UniFFI Translation Layer**: Bridges the Rust core and frontends, enabling cross-platform compatibility [24].
- **iOS Frontend** (Swift): Provides user interface for device management, data visualization, and notifications.

Figure 2 provides a high-level overview of VitalVision's architecture, illustrating the relationships and data flow between its components.

The following subsections provide detailed descriptions of the Core Library and iOS Frontend components, explaining their implementation and interactions.

Core Library

The core library of VitalVision is implemented in Rust, a decision based on several important technical considerations:

- **Cross-platform Compatibility**: Rust's inherent ability to compile to native binaries allows for seamless usage of the core library on both iOS and Android platforms [25]. This capability ensures consistent behaviour and performance across both platforms with minimal required code adaptation.
- **Performance and Memory Safety**: Rust's architecture provides a good balance between performance and safety. Its zero-cost abstractions allow for efficient code execution while maintaining memory safety through features such as the ownership system and borrowing rules [24].
- **Crate Ecosystem**: Rust offers a rich ecosystem of libraries (known as crates), with the official Rust package registry hosting over 150,000 crates as of 2024 [26]. The specific libraries used in this project include `btleplug` for platform-abstraced BLE communication [27], `ndarray` for efficient data handling [28], `Tokio` for asynchronous execution [29], and as established before, `UniFFI` for generating language bindings [30].

Note: While Rust's ecosystem for digital signal processing is growing, we found it to be not quite as mature as in some other programming languages at the time of development. For VitalVision, we addressed this by implementing standard DSP algorithms using lower-level components already available in Rust. For instance, we created our own Butterworth filters using existing biquad filter implementations. These custom implementations and their integration into our analysis pipeline are discussed in more detail in section 4.

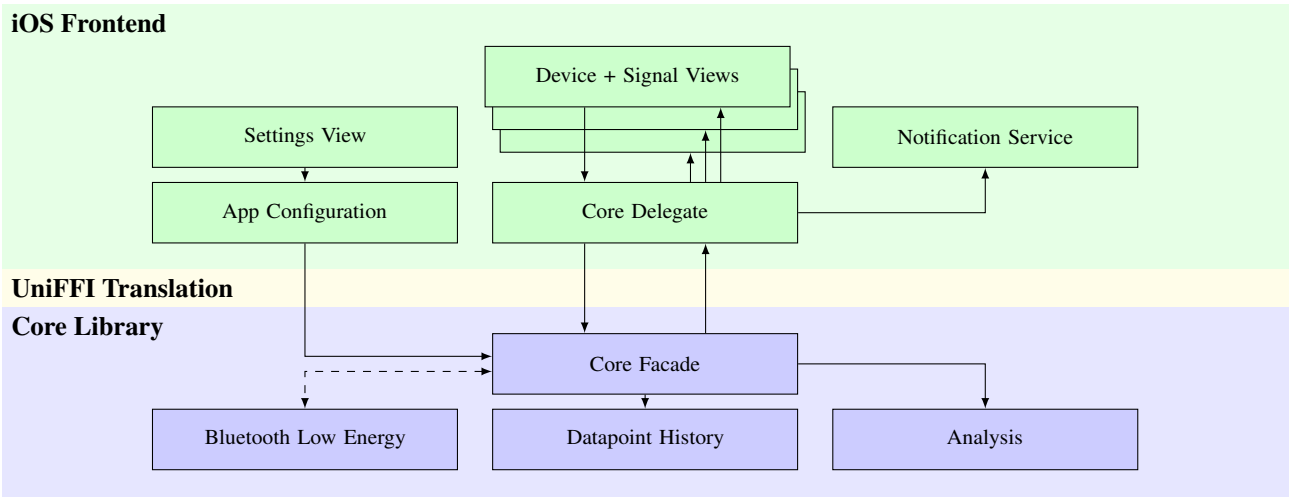


Figure 2: The System Architecture of VitalVision. It is organized in three layers: the iOS Frontend (green), UniFFI Translation (yellow), and Core Library (blue). Components within each layer are shown as boxes, and arrows indicate data flow, highlighting bidirectional communication between components and through layers. This includes device changes and data points moving from the core to the frontend, as well as user interactions and configuration changes in the frontend triggering actions in the core.

Core Facade Component

The Core Facade component serves as the primary API for interacting with the VitalVision core library. It simplifies frontend development by managing internal communication and presenting a streamlined interface. It is instantiated with a configuration object that allows fine-tuning of various parameters such as analysis window sizes, update frequencies, and signal quality thresholds. For a complete reference of the API's structures and methods, see Appendix A.

The API uses a delegate pattern to communicate updates to the frontend through two methods. The `devices_changed` method is called when there's a change in the device list or any device's status, providing an updated list of all devices. A device represents a wearable with its identifier, status, battery level, and a list of channels (like ECG or PPG). The `new_data` method is called when new data points are received for a specific channel, passing the channel identifier and latest data. This approach separates device management from data streaming, enabling efficient visualization in the frontend.

The API also offers methods for managing Bluetooth Low Energy (BLE) connections, including functions to start device discovery, pause all connections (useful for battery conservation), and resume connections. It also provides a method to trigger time synchronization between the smartphone and connected devices manually.

All structures and methods of the Core Facade are declared as UniFFI exports in the `uniffi.udl` file. This Universal Definition Language (UDL) file specifies the API surface that will be exposed to other languages. A custom build command in the project's build script uses the uniffi-bindgen tool to generate language bindings from this UDL file. These bindings are essentially header files for the binary, created in a separate folder. This process allows the Rust core library to be used as if it were a native library in the target language.

Bluetooth Low Energy Component

The BLE component handles device discovery, connection management, and data transmission. It also provides an interface for resynchronizing device clocks, pausing, and resuming connections, corresponding to the respective functions in the core API. This component is initiated by the `start_ble_loop` call in the core interface.

Concurrent handling of multiple devices requires a multi-queue architecture, which uses Tokio tasks for scheduling and channels for inter-queue communication[29]. For cross-platform compatibilty, the component uses btleplug, which provides a unified abstraction layer for BLE operations across operating systems[27].

The connection process follows a structured sequence: First, the BLE adapter initiates device discovery. Upon device connection, services and characteristics are iterated to retrieve device information such as name, serial, type, and battery level. To initiate data capture, the current timestamp is set on the device and subsequently read back to calculate the clock drift. Then, a scheduled task for periodic timestamp resynchronization is created. The component subscribes to the battery service for power monitoring and to the data service, which provides the actual signal data in the form of BLE update messages. Finally, a notification handler is set up to parse incoming data and forward it to the datapoint history component.

For SIP VitalTracker devices, data updates occur every 100 milliseconds, and each data update contains a compact, logically structured blob. It starts with a packet counter byte, and following this, there are three sequences of data, each representing a snapshot of the device's sensors. A sequence consists of 2 bytes for ECG data (or zeros if ECG isn't available) and 6 bytes for PPG data (2 bytes each for green, red, and infrared channels).

Datapoint History Component

The datapoint history of VitalVision's core library manages incoming datapoints and stores them for visualization and analysis. While datapoints are received individually from the BLE component, they need to be visualized and analyzed as a continuous history. To address this, the datapoint history component implements a specialized ring buffer.

It is implemented as a sliceable, double-backed ring buffer structure to allow for efficient insertion of new datapoints without shifting existing data, as well as retrieval of contiguous slices of recent datapoints without internal copying. The buffer's size is set to the larger of `hist_size_api` and `hist_size_analysis` (see Appendix A), used for both API callbacks and internal analysis processes.

Note: While this implementation minimizes copying within the core library, data must still be copied when passing through the UniFFI layer to the frontend, an unavoidable consequence of crossing the language boundary.

Note: A ring buffer backed by double mapped virtual memory was also considered, but memory mapping is not easily possible in the iOS sandboxed process environment.

Analysis Component

The analysis component is responsible for processing the stored datapoints to assess signal quality in real-time. It implements the algorithms for ECG and PPG quality assessment and heart rate estimation, described in detail in section 4.

iOS Frontend

The VitalVision iOS frontend serves as the user interface for device management and data visualizaton. It provides an intuitive interface for researchers to monitor and analyze data from multiple devices simultaneously.

The VitalVision iOS frontend is built using Swift and SwiftUI. Latter provides a declarative approach to UI development [31], allowing for efficient creation of responsive interfaces. For signal visualization, the app uses Swift Charts, which offers performant and customizable data plots [32].

To receive updates from the core, the iOS app implements the VVCoreDelegate protocol defined by the core library. Integration between the Swift host application and the Rust core library is achieved through the UniFFI-generated language bindings. A custom build script, `build-ios.sh`, is used to build the core library for iOS. This script runs the necessary build commands and bundles the compiled binary together with the UniFFI-generated headers into a `.framework` file. This .framework can then be imported and used as a normal Swift library in the iOS project. The generated bindings map Rust-defined types to their Swift equivalents, effectively exposing the core library as a native Swift API.

App Navigation

The VitalVision app is structured around three main screens, as seen in Figure 3: the device overview, device details, and channel details.

The device overview serves as the app's main screen and entry point. It presents a scrollable list of all connected devices, with each list item presenting a brief device summary:

- Device name and identifier
- Battery level, represented as a percentage and color-coded icon
- Time drift in milliseconds, with an option to initiate resynchronization
- Overall signal quality assessment, prominently displayed using color-coded indicators

The list of devices can be sorted based on various criteria such as participant ID or device location, which can be set per device. This sorting functionality is particularly useful in multi-participant studies or when managing a large number of devices.

At the top of the screen, users also find a button for resyncing the time of devices, and a button to disconnect and reconnect all devices for battery conservation. A settings button provides access to app-wide configuration options.

Tapping on a device in the list navigates to the device details screen, providing a more comprehensive view of the selected device. It offers editable fields for the aforementioned participant ID and device location, and presents a list of all channels associated with the device. Each channel in the list displays its name, type (ECG or PPG), and current signal quality. Users can designate a channel as "watched" through a long-press gesture, which integrates with the app's notification service for targeted quality alerts. A small bell icon appears next to watched channels for easy identification.

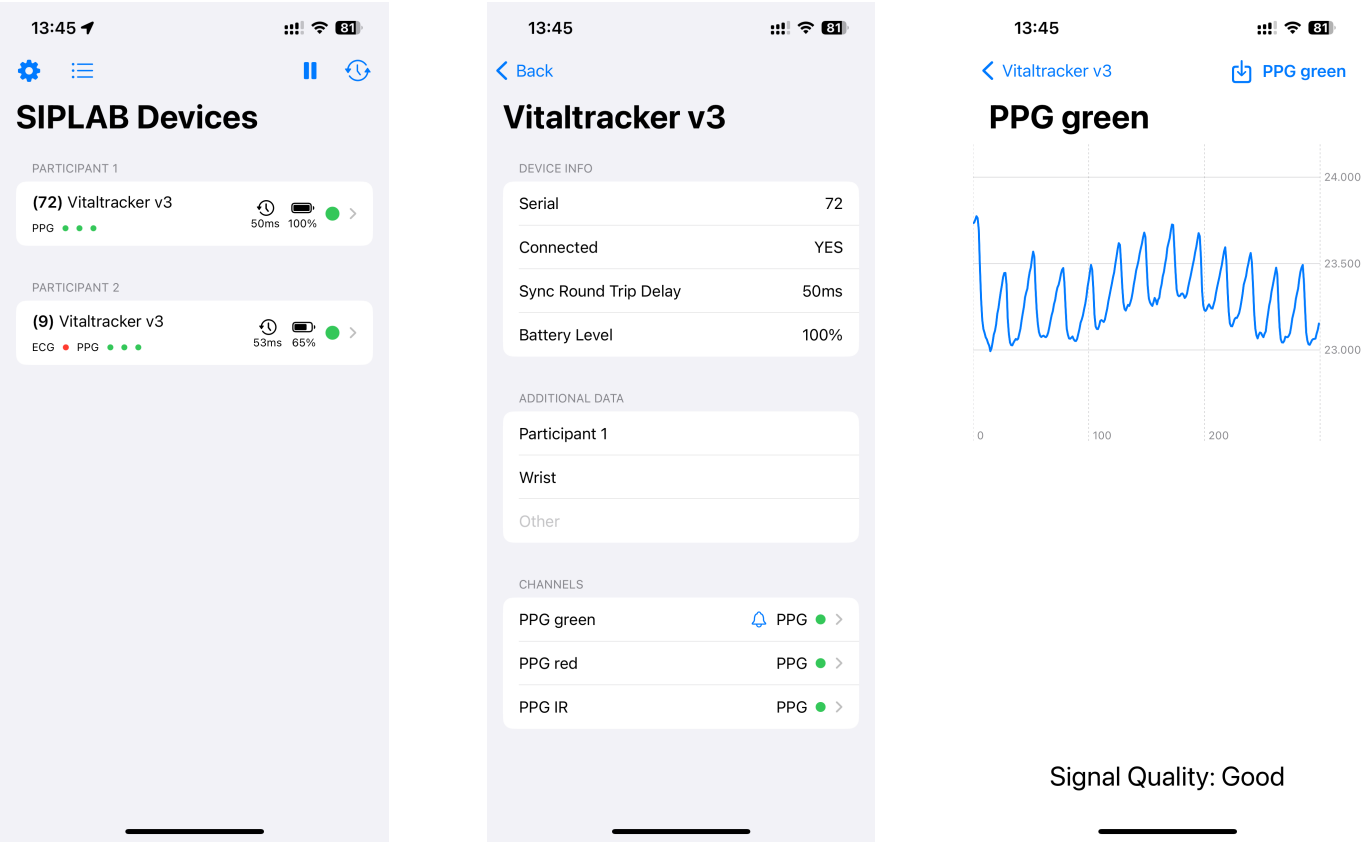
Selecting a channel from this list navigates to the channel details screen. This screen is dominated by a large, real-time plot of the incoming data stream, implemented using Swift Charts. The chart adapts its display based on the channel type, showing appropriate y-axis scales for ECG or PPG data. For development and debugging purposes, the current trace can be exported to the device, allowing developers to analyze the data offline or use it for further testing.

Settings

The settings screen allows users to configure all parameters of the core configuration section described in Appendix A. This includes adjusting analysis window sizes, update frequencies, signal quality thresholds, and all analysis parameters. It provides a user-friendly interface for fine-tuning the app's behavior to meet specific research requirements.

Notification Service

The notification service operates continuously, even when the app runs in the background. It subscribes to signal quality reports from the core library for all channels marked as watched, comparing quality against a predefined threshold. If quality falls below the threshold for a specified duration, the service triggers a local notification on the device's lock screen. The notification includes the device identifier, channel name, and a brief description of the quality issue.



(a) (b) (c)

Figure 3: VitalVision App Interface: (a) Device Overview screen listing connected devices, (b) Device Details screen showing specific device information and channels, (c) Channel Data View displaying real-time PPG signal and quality assessment.

ANALYSIS ALGORITHMS

As established in Section 2, our algorithms adapt existing techniques for efficient real-time signal quality assessment on mobile devices. The following subsections detail the specific implementations of these algorithms.

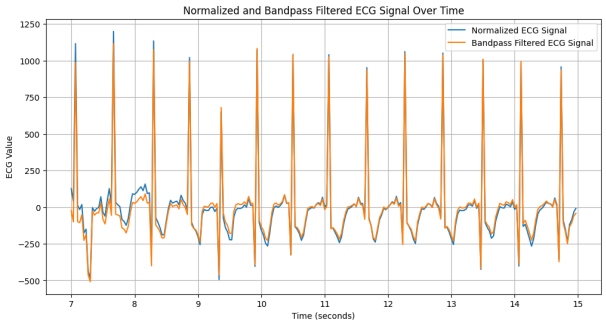
ECG Algorithm

For ECG analysis, the central idea revolves around the characteristic R peaks in the cardiac cycle. The time intervals between these R peaks, known as R-R intervals, form the basis of our quality assessment. In a normal ECG signal, these intervals should fall within a physiologically plausible range, typically corresponding to heart rates between 40 and 200 beats per minute (bpm). Sudden, drastic changes in these intervals are unlikely in typical physiological conditions. Therefore, by examining the consistency and plausibility of these R-R intervals, we can infer the quality of the ECG signal. Intervals that fall outside the expected range or change too rapidly between consecutive beats likely indicate signal disturbances rather than true cardiac events.

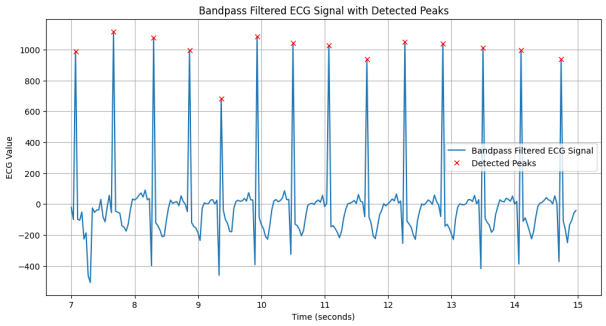
- Subtract the mean from the raw signal to center it around zero.
- Apply a highpass filter to remove baseline wander and low-frequency noise.
- Compute the Median Absoulte Deviation (MAD) of the signal as a robust measure of variability.
- Use the MAD as an adaptive threshold for peak detection.
- Apply a peak finding algorithm with specific parameters to identify potential R peaks.
- Compute the time intervals between consecutive R peaks.
- Convert inter-peak intervals to instantaneous heart rates in bpm.
- Apply range checks to filter out physiologically implausible heart rates (e.g., 40-200 bpm).
- Evaluate the rate of change between consecutive heart rate estimates, filtering out rapid fluctuations that exceed physiological limits.
- Calculate the proportion of valid heart rate estimates within the expected range.

Figure 4 illustrates the key steps of our ECG analysis algorithm: signal normalization and filtering, pulse detection, and quality assessment.

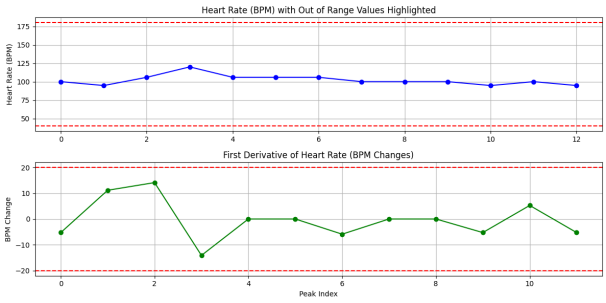
In the implementation, we utilize a Butterworth highpass filter, implemented using biquad filters, to remove baseline wander



(a)



(b)



(c)

Figure 4: ECG Algorithm: (a) Signal Normalization and Filtering (b) Pulse Detection (c) Quality Assessment

and low-frequency noise. This filter type was chosen for its maximally flat frequency response in the passband, which helps preserve the shape of the ECG signal. To minimize phase distortion and ensure zero-phase filtering, we apply the filter in both forward and backward direction. This technique, known as forward-backward filtering, effectively doubles the order of the filter while maintaining zero phase shift. For peak detection, we used an existing Rust reimplementation of the core functionality of SciPy's `find_peaks` algorithm. This algorithm primarily relies on peak prominence, which measures how much a peak stands out due to its intrinsic height and location relative to other peaks. We set the minimum prominence as a multiple of the MAD, providing an adaptive threshold that adjusts to the signal's variability. Additionally, we apply a minimum distance between peaks to prevent detecting multiple peaks within a single cardiac cycle, and a maximum plateau size to discard peaks that are too wide to be valid R peaks.

Considering the sampling frequency, the ECG signal is initially sampled at 128 Hz on the device and then downsampled to 30 Hz using a custom decimation filter with adaptive alignment, before being streamed to the app. This approach balances the energy cost of BLE communication with sufficient temporal resolution for accurate R peak detection in most scenarios.

The algorithm's parameters were specifically tuned for SIPLAB VitalTracker devices using a hand-labeled development dataset. Table 1 lists the optimal thresholds obtained by this process.

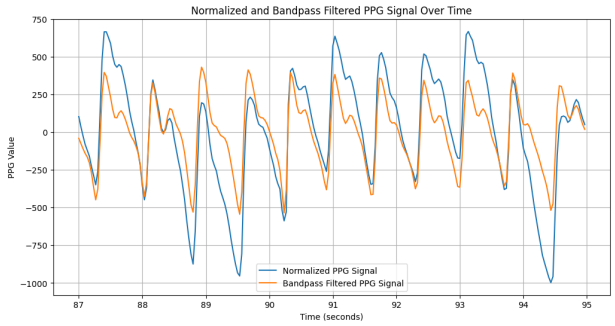
Parameter	Value
Sampling frequency	30.0 Hz
Highpass filter cutoff frequency	0.6 Hz
Filter order	1
R peak prominence MAD multiple	12.0
Minimum distance between R peaks	10 samples
Maximum R peak plateau size	3 samples
Heart rate lower bound	40.0 bpm
Heart rate upper bound	200.0 bpm
Maximum allowed difference between consecutive heart rate estimates	20.0 bpm

Table 1: ECG Analysis Algorithm Parameters for SIPLAB VitalTracker Devices

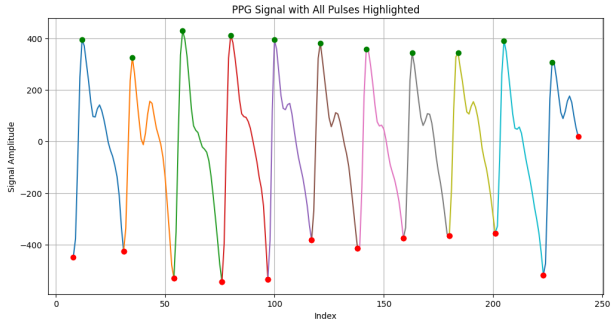
PPG Algorithm

The PPG analysis takes a different approach, focusing on the morphology of individual pulse waves. Each pulse in a PPG signal represents a cardiac cycle and should exhibit certain characteristics in a clean signal. We examine three key aspects of each pulse: its amplitude, the difference in height between consecutive trough points, and the pulse width. For a given PPG sensor, pulse amplitudes should fall within a certain range, neither too small (indicating weak signal) nor too large (suggesting potential sensor saturation). The difference in height between consecutive troughs, when compared to the pulse amplitude, can reveal drastic baseline shifts or motion artifacts. Pulse width, representing the time between consecutive troughs, should align with physiologically plausible heart rates. By applying thresholds to these characteristics, we can distinguish between clean pulses and those likely corrupted by artifacts.

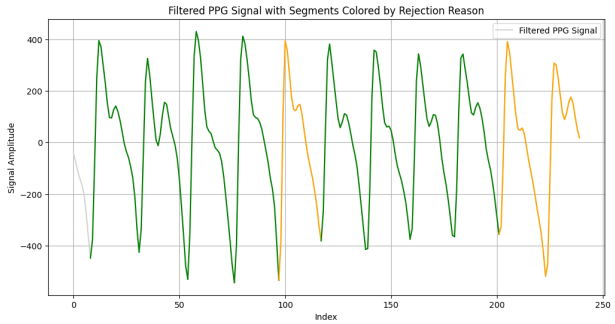
- Subtract the mean from the raw signal to center it around zero.
- Apply a bandpass filter to remove high-frequency noise and low-frequency trends.
- Estimate the lower envelope of the signal to identify potential pulse troughs.
- Locate pulse troughs by finding points where the signal intersects with the lower envelope.
- Identify the maximum point between consecutive troughs as pulse peaks.
- Define individual pulses as the signal segments between consecutive troughs.



(a)



(b)



(c)

Figure 5: PPG Algorithm: (a) Signal Normalization and Filtering (b) Pulse Detection (c) Quality Assessment

- Compute the amplitude of each pulse from trough to peak.
- Calculate the height difference between consecutive troughs.
- Measure the time duration of each pulse.
- Apply thresholds to validate pulse amplitudes, trough depth differences relative to pulse amplitudes, and pulse widths.
- Compute the average width of all valid pulses.
- Calculate the proportion of valid pulses relative to the expected number of pulses for the signal duration and average width.

Figure 5 visualizes these steps of the PPG analysis algorithm.

Note: For illustration purposes, the PPG signals in this Figure are inverted to match common literature representations. In the VitalVision app, the signal is handled and displayed "raw" as delivered from the wearables, effectively appearing upside-down compared to this figure. This difference in representation does not affect the algorithm's functionality or accuracy.

In the implementation, we utilize a Butterworth bandpass filter, also implemented using biquad filters, to remove both high-frequency noise and low-frequency trends. As with the ECG filter, we apply forward-backward filtering to ensure zero phase shift. This preserves the fundamental shape of the PPG waveform while eliminating unwanted signal components.

For pulse segmentation, we employ a lower envelope estimation technique inspired by Sukor et al. [20]. While an alternative approach using a peak finding algorithm to detect troughs (similar to the method used in ECG analysis) was considered, the envelope estimation technique proved sufficiently effective for our application and was therefore implemented.

To avoid situations where the signal quality is falsely inflated due to detecting too few pulses, we set a lower bound on the expected number of pulses based on the calculated average valid pulse width. This approach helps maintain robustness in noisy signal conditions.

As for the ECG case, this algorithm's parameters were specifically tuned for SIPLAB VitalTracker devices, resulting in the optimal thresholds shown in Table 2.

Parameter	**Value**
Sampling frequency	30.0 Hz
Bandpass filter low cutoff frequency	1.0 Hz
Bandpass filter high cutoff frequency	10.0 Hz
Filter order	4
Lower envelope estimation window	23 samples
Minimum pulse amplitude	10 LSBs
Maximum pulse amplitude	2000 LSBs
Minimum trough depth difference	-25% of amplitude
Maximum trough depth difference	25% of amplitude
Minimum pulse width	0.333 seconds
Maximum pulse width	1.5 seconds

Table 2: PPG Analysis Algorithm Parameters for SIPLAB VitalTracker Devices

EVALUATION METHOD

To evaluate our ECG and PPG analysis algorithms, we used a preexisting SIPLAB dataset. This dataset comprised ECG and PPG recordings from 3 participants, each spanning approximately 13 hours of everyday activities. Data was collected from 4 devices placed at different body locations: chest (ECG and PPG), head (PPG), ankle (PPG), and wrist (PPG).

Given the sensitive nature of physiological data, all participant data was anonymized and handled in compliance with ethical guidelines to ensure privacy and confidentiality throughout the evaluation process. Due to the confidentiality agreements in place, the specific dataset used for this evaluation is not publicly available in the project repository. Researchers interested

in replicating or extending this work are encouraged to contact the Sensing, Interaction and Perception Lab at ETH Zürich for guidance on accessing similar datasets or for potential collaboration.

Dataset Preparation

The original data was recorded at 128 Hz. We downsampled it to 30 Hz using a custom decimation filter, mirroring the process used on the wearable devices for BLE streaming. This step aligned our evaluation data with the data processed by the VitalVision application.

From this processed dataset, we extracted 50 random traces for each participant, resulting in 150 traces for each signal type (ECG and PPG). For PPG, we included traces from various body locations and different color channels.

Each trace consisted of 500 data points, representing approximately 16 seconds of recording. This trace length was chosen to balance between capturing sufficient signal trends and maintaining sensitivity to localized disturbances, while also aligning with typical analysis windows used in real-time physiological monitoring applications [17].

Manual Classification

To establish a ground truth, each trace was manually classified. For ECG traces, classification was based on the visibility of R peaks:

- **Good**: Clearly visible and regular R peaks
- **Poor**: R peaks not consistently identifiable

The PPG classification used a three-tier system:

- **Good**: Clean pulses, low noise, minimal baseline drift, no motion artifacts
- **Fair**: Noticeable noise but recognizable pulses, or presence of a single motion artifact in an otherwise clean trace
- **Poor**: Excessive noise or artifacts, pulses not clearly recognizable throughout the trace

To aid in PPG classification, we correlated the PPG signals with the corresponding ECG traces, which could be inspected side-by-side during the labeling process. While this correlation aided our manual classification, our algorithm assesses PPG quality independently of ECG data.

Execution

We executed our analysis algorithms on each trace using Python bindings for our core library. These bindings were generated using a custom `build-py.sh` script, similar to the process used for iOS bindings. This approach ensured consistency between our app's real-time analysis and our offline evaluation. The algorithms were run with the parameters specified in Tables 1 and 2. For each trace, we recorded the quality score produced by our algorithm, representing the proportion of valid R-R intervals for ECG traces and valid pulses for PPG traces.

RESULTS

ECG Algorithm

Our ECG algorithm demonstrated strong performance in distinguishing between good quality signals and those without a discernible trace. The dataset consisted of 150 traces, with 127 (84.67%) classified as good quality (Class 0) and 23 (15.33%) as poor quality (Class 2). For the used dataset, the algorithm achieved perfect classification:

- **Overall Accuracy**: 1.0000
- **Precision, Recall, and F1-score** for both classes: 1.0000

Figure 6a shows the confusion matrix for the ECG algorithm. The example traces in Figure 7 illustrate the algorithm's performance across different ECG samples, correctly identifying both good and poor quality ECG signals.

PPG Algorithm

The PPG algorithm faced a more complex classification task compared to the ECG algorithm, since the greater variability in PPG waveform morphology necessitates the use of three quality categories instead of two. The dataset again comprised 150 traces, with 35 (23.33%) classified as good quality (Class 0), 44 (29.33%) as fair quality (Class 1) and 71 (47.33%) as poor quality (Class 2)

The algorithm's performance varied across classes:

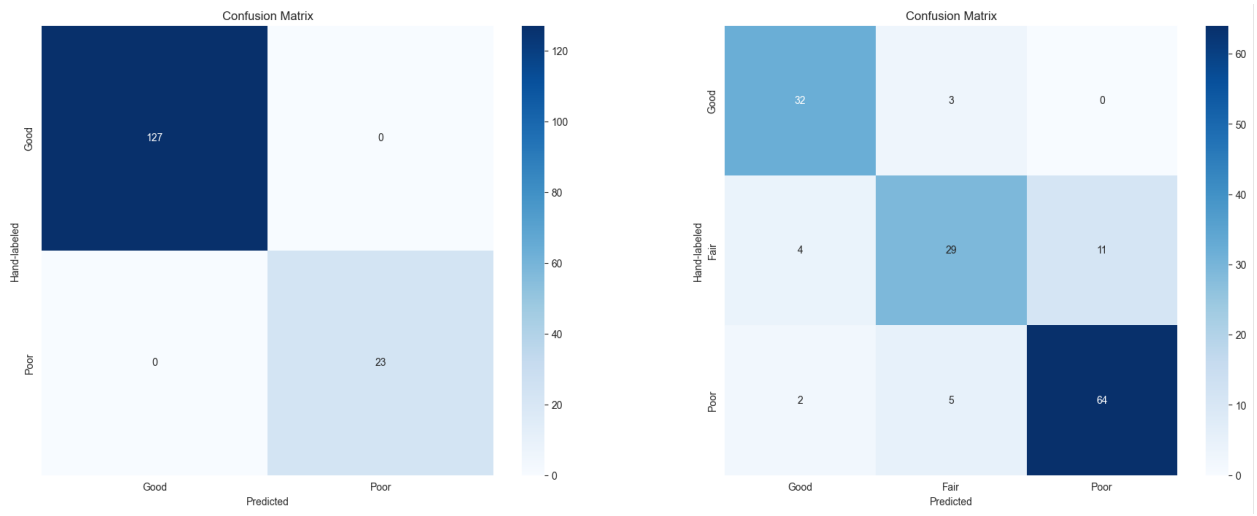
- **Overall Accuracy**: 0.8333
- **Class 0** (Good): Precision: 0.8421, Recall: 0.9143, F1-score: 0.8767
- **Class 1** (Fair): Precision: 0.7838, Recall: 0.6591, F1-score: 0.7160
- **Class 2** (Poor): Precision: 0.8533, Recall: 0.9014, F1-score: 0.8767

Figure 6b shows the confusion matrix for the PPG algorithm. The example traces in Figure 8 illustrate the algorithm's performance across different PPG samples, correctly identifying clearly good and clearly poor signals, while also facing its challenges with borderline cases and certain types of noise and artifacts.

DISCUSSION

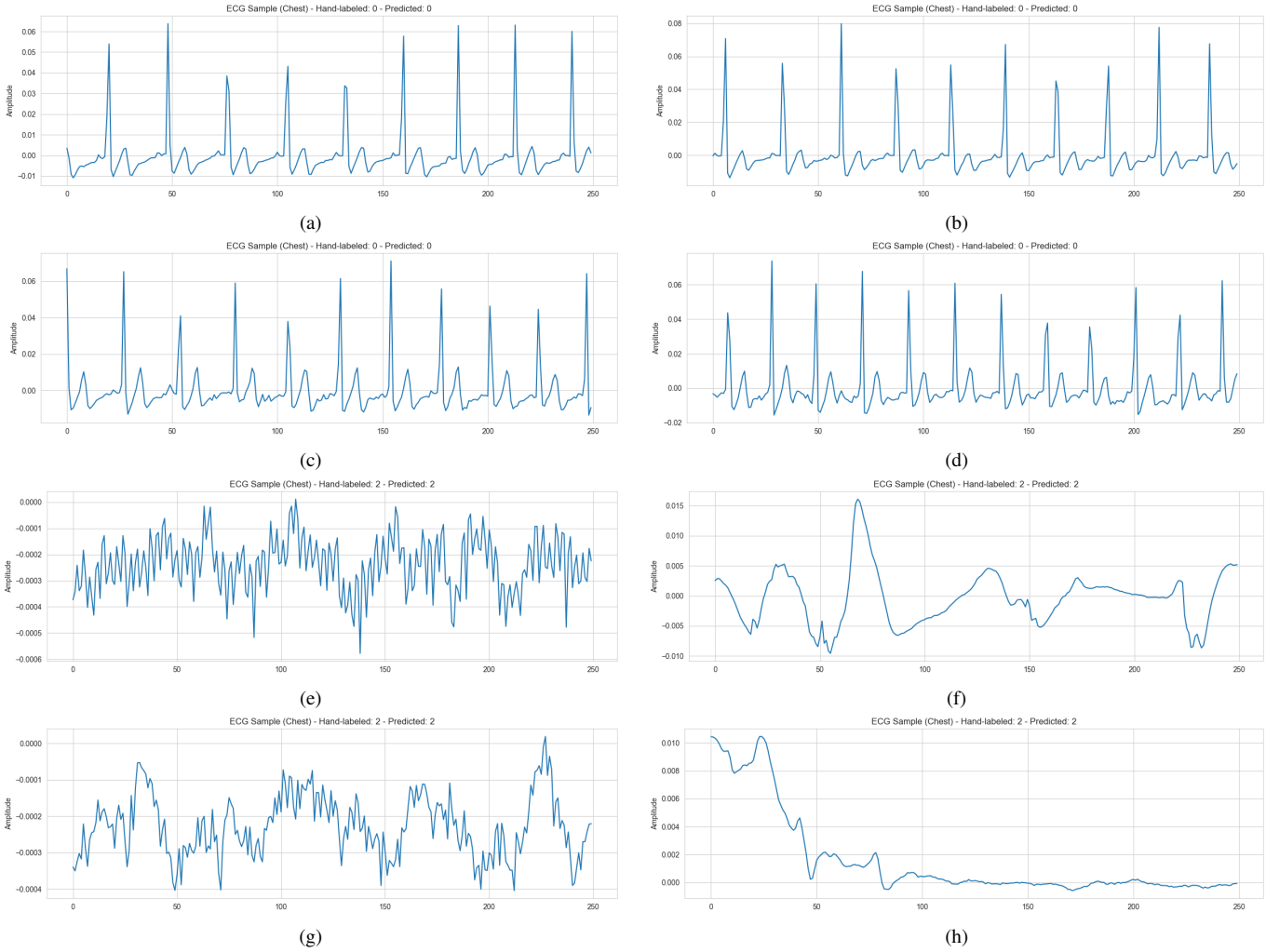
The ECG algorithm achieved perfect classification accuracy in distinguishing between good quality signals with clearly visible R peaks and poor quality signals without discernible patterns. However, it's important to note that this perfect performance may partly reflect the relatively clear-cut nature of our dataset, where traces were either clearly good or clearly unusable. In more nuanced scenarios, such as those with intermittent noise or subtle artifacts, the algorithm's performance might vary. This requires further investigation.

The PPG algorithm showed strong performance in identifying both good and poor quality signals, with F1-scores of 0.8767 for both classes. It demonstrated slightly lower performance in identifying fair quality signals; it tends towards conservative quality assessments, sometimes classifying fair quality signals



(a) (b)

Figure 6: Confusion Matrices for the ECG (a) and PPG (b) algorithms



(a) (b) (c) (d) (e) (f) (g) (h)

Figure 7: Example ECG Traces: (a)-(d): Good quality ECG signals with clearly visible R peaks, correctly identified by the algorithm. (e)-(h): Traces without discernible ECG patterns, correctly classified as poor quality.

Figure 8: Example PPG Traces: (a)-(b): Correctly identified as good quality traces. (c): Trace with baseline drift that can be filtered away, as the pulse shape remains clearly visible. (d): Classified as "fair" due to some noise and intermediate pulses. (e): Traces containing baseline drift that affects peak height and is difficult to filter away. (f)-(g): Pure noise correctly identified as poor quality. (h): Up and down movement too wide for a pulse, correctly classified as poor. (i): Up and down movement without recognizable pulses, but the algorithm incorrectly identifies some pulses, classifying it as fair. (j): Noisy signal misclassified as good by the algorithm. (k)-(l): When correlated with ECG, pulses can be recognized (would be fair quality), but the algorithm incorrectly rejects these as poor.

as poor. This cautiousness, while reducing false positives, may lead to overreporting of quality issues. Overall, the algorithm achieved 83.33 % classification accuracy. Despite these limitation, the algorithm remains effective for alerting purposes, reliably flagging potentially problematic data segments.

To enhance its performance, we propose implementing self-similarity analysis of waveforms as described in related work. This approach, combined with more lenient thresholds, could improve discrimination between fair and poor quality signals. By analyzing the consistency of pulse shapes within a signal segment, we may better distinguish between minor artifacts and significant distortions.

In summary, both algorithms seem suitable for real-time signal quality assessment in wearable physiological monitoring, each with distinct characteristics. While the current implementation provides a good foundation, further refinement and testing with diverse, expertly labeled datasets are necessary to establish reliability across a wider range of real-world conditions.

Future Work

To expand the system's capabilities and address current limitations, we propose the following:

1. Develop an Android frontend: Due to time constraints, this component could not be implemented during the thesis period. An Android version would significantly expand VitalVision's accessibility, making it a top priority for future development.
2. Enhance the PPG algorithm based on the suggested improvements, focusing on better differentiation between fair and poor quality signals.
3. Collect more diverse data with expert labeling to replace the current self-labeled dataset. This would provide a more robust ground truth for algorithm evaluation and refinement.
4. Incorporate additional sensors (e.g., accelerometers) into the evaluation process to better contextualize signal quality variations and potentially distinguish between physiological changes and motion artifacts.

CONCLUSION

This thesis presented VitalVision, a mobile platform for real-time validation of ECG and PPG data from wearable devices. The system comprises algorithms implemented in a Rust-based core library, enabling detection of signal disruptions and cross-platform data processing. An iOS frontend provides multi-device monitoring and alert management.

By shifting signal validation to a smartphone, VitalVision enables continuous quality monitoring during extended studies while preserving wearable device battery life. This approach addresses a key challenge in wearable health monitoring: maintaining data quality in dynamic conditions during long-term data collection.

Our evaluation demonstrated the ECG algorithm's effectiveness in distinguishing between clear signals and unusable data, while the PPG algorithm showed promise in identifying good quality signals but tended towards conservative assessments. These results suggest potential utility for real-time quality assessment in physiological monitoring studies, though further refinement and testing are needed.

In conclusion, VitalVision represents a step forward in addressing challenges in wearable health monitoring, particularly for long-term in-the-wild studies. While the current implementation is promising, there are opportunities for enhancement and expansion. Future developments, including an Android frontend and refined algorithms, have the potential to further improve the platform's utility and accessibility.

As research in wearable health monitoring continues to evolve, systems like VitalVision can play an important role in enabling more reliable and comprehensive physiological data collection across diverse real-world scenarios.

Acknowledgements

I would like to express my sincere gratitude to Prof. Dr. Christian Holz and the Sensing, Interaction and Perception Lab at ETH Zürich for their support, infrastructure, and research environment.

I owe a special debt of gratitude to my thesis supervisor, Manuel Meier. His support and technical know-how were absolutely essential from start to finish. Manuel didn't just guide my work - he really helped shape it into something I'm proud of. Beyond the technical aspects, I'm particularly grateful for his help in bridging my connection with ETH and guiding me through the difficulties of being an external student. I deeply appreciate the trust he placed in me to undertake this project, and I feel very fortunate to have had him as my supervisor.

I would also like to extend my sincere thanks to my local supervisor at Paris Lodron University Salzburg, Assoc. Prof. Dr. Andreas Naderlinger. His assistance with the administrative procedures for conducting an external bachelor's thesis was invaluable, enabling me to pursue this opportunity. Furthermore, I am grateful for his extensive feedback throughout the thesis, which significantly elevated the quality of this work.

Furthermore, I would like to acknowledge all the other members of the SIPLAB for fostering a friendly and collaborative atmosphere, which greatly enhanced my research experience.

REFERENCES

[1] World Health Organisation. 2021. Cardiovascular diseases (CVDs). (06 2021). https://www.who.int/news-room/fact-sheets/detail/cardiovascular-diseases-(cvds)

[2] Sophie Huhn, Miriam Axt, Hanns-Christian Gunga, Martina Anna Maggioni, Stephen Munga, David Obor, Ali Sié, Valentin Boudo, Aditi Bunker, Rainer Sauerborn, Till Bärnighausen, and Sandra Barteit. 2022. The Impact of Wearable Technologies in Health Research: Scoping Review. *JMIR mHealth and uHealth* 10, 1 (01 2022), e34384. DOI: http://dx.doi.org/10.2196/34384

[3] Mohammad Ghamari, Denisse Castaneda, Aibhlin Esparza, Cinna Soltanpur, and Homer Nazeran. 2018. A

review on wearable photoplethysmography sensors and their potential future applications in health care. *International Journal of Biosensors and Bioelectronics* 4, 4 (2018). DOI: http://dx.doi.org/10.15406/ijbsbe.2018.04.00125

[4] Federica Laricchia. 2024. Statistics and Facts on Wearable Technology | Statista. (07 2024). https://www.statista.com/topics/1556/wearable-technology

[5] Yongbo Liang, Zhencheng Chen, Rabab Ward, and Mohamed Elgendi. 2018. Hypertension Assessment via ECG and PPG Signals: An Evaluation Using MIMIC Database. *Diagnostics* 8, 3 (Sept. 2018), 65. DOI: http://dx.doi.org/10.3390/diagnostics8030065

[6] Neha, H. K. Sardana, R. Kanwade, and S. Tewary. 2021. Arrhythmia detection and classification using ECG and PPG techniques: a review. *Physical and Engineering Sciences in Medicine* 44, 4 (01 Dec 2021), 1027–1048. DOI:http://dx.doi.org/10.1007/s13246-021-01072-5

[7] Mahmoud M. Bassiouni, Islam Hegazy, Nouhad Rizk, Sayed A. El-Dahshan, and Abdelbadeeh M. Salem. 2021. Combination of ECG and PPG Signals for Healthcare Applications: A Survey. *Advances in Modelling and Analysis* 64 (Dec. 2021). DOI: http://dx.doi.org/10.18280/ama_b.641-409

[8] Seyed Mohamad Amin Salehizadeh. 2015. Motion and Noise Artifact Detection and Vital Signal Reconstruction in ECG/PPG based Wearable Devices. *Doctoral Dissertations* (2015). https://digitalcommons.lib.uconn.edu/dissertations/980

[9] Saifur Rahman, Chandan Karmakar, Iynkaran Natgunanathan, John Yearwood, and Marimuthu Palaniswami. 2022. Robustness of electrocardiogram signal quality indices. *Journal of The Royal Society Interface* 19, 189 (April 2022). DOI: http://dx.doi.org/10.1098/rsif.2022.0012

[10] Chathuri Daluwatte, Lars Johannesen, Loriano Galeotti, Jose Vicente, David G. Strauss, and Christopher G. Scully. 2016. Assessing ECG signal quality indices to discriminate ECGs with artefacts from pathologically different arrhythmic ECGs. *Physiological Measurement* 37, 8 (July 2016). DOI: http://dx.doi.org/10.1088/0967-3334/37/8/1370

[11] Jiapu Pan and Willis J. Tompkins. 1985. A Real-Time QRS Detection Algorithm. *IEEE Transactions on Biomedical Engineering* BME-32, 3 (1985), 230–236. DOI:http://dx.doi.org/10.1109/TBME.1985.325532

[12] Luca Neri, Matt T. Oberdier, Antonio Augello, Masahito Suzuki, Ethan Tumarkin, Sujai Jaipalli, Gian Angelo Geminiani, Henry R. Halperin, and Claudio Borghi. 2023. Algorithm for Mobile Platform-Based Real-Time QRS Detection. *Sensors* 23, 3 (2 2023). DOI: http://dx.doi.org/10.3390/s23031625

[13] Luca Neri, Ilaria Gallelli, Massimo Dall'Olio, Jessica Lago, Claudio Borghi, Igor Diemberger, and Ivan Corazza. 2024. Validation of a New and Straightforward Algorithm to Evaluate Signal Quality during ECG Monitoring with Wearable Devices Used in a Clinical Setting. *Bioengineering* 11, 3 (Feb. 2024), 222. DOI: http://dx.doi.org/10.3390/bioengineering11030222

[14] Kirina van der Bijl, Mohamed Elgendi, and Carlo Menon. 2022. Automatic ECG Quality Assessment Techniques: A Systematic Review. *Diagnostics* 12, 11 (Oct. 2022), 2578. DOI: http://dx.doi.org/10.3390/diagnostics12112578

[15] Liping Xie, Zilong Li, Yihan Zhou, Yiliu He, and Jiaxin Zhu. 2020. Computational diagnostic techniques for electrocardiogram signal analysis. *Sensors* 20, 21 (2020), 6318.

[16] Guoyang Liu, Xiao Han, Lan Tian, Weidong Zhou, and Hui Liu. 2021. ECG quality assessment based on hand-crafted statistics and deep-learned S-transform spectrogram features. *Computer Methods and Programs in Biomedicine* 208 (2021), 106269. DOI: http://dx.doi.org/https://doi.org/10.1016/j.cmpb.2021.106269

[17] Serena Moscato, Stella Lo Giudice, Giulia Massaro, and Lorenzo Chiari. 2022. Wrist Photoplethysmography Signal Quality Assessment for Reliable Heart Rate Estimate and Morphological Analysis. *Sensors* 22, 15 (Aug. 2022), 5831. DOI: http://dx.doi.org/10.3390/s22155831

[18] Gabriele B. Papini, Pedro Fonseca, Xavier L. Aubert, Sebastiaan Overeem, Jan W.M. Bergmans, and Rik Vullings. 2017. Photoplethysmography beat detection and pulse morphology quality assessment for signal reliability estimation. In *2017 39th Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC)*. 117–120. DOI: http://dx.doi.org/10.1109/EMBC.2017.8036776

[19] Dae-Geun Jang, Ui Kun Kwon, Seung Keun Yoon, Changsoon Park, Yunseo Ku, Seung Woo Noh, and Youn Ho Kim. 2018. A Simple and Robust Method for Determining the Quality of Cardiovascular Signals Using the Signal Similarity. In *2018 40th Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC)*. 478–481. DOI: http://dx.doi.org/10.1109/EMBC.2018.8512341

[20] J Abdul Sukor, S J Redmond, and N H Lovell. 2011. Signal quality measures for pulse oximetry through waveform morphology analysis. *Physiological Measurement* 32, 3 (Feb. 2011). DOI: http://dx.doi.org/10.1088/0967-3334/32/3/008

[21] Mohamed Elgendi. 2016. Optimal Signal Quality Index for Photoplethysmogram Signals. *Bioengineering* 3, 4 (Sept. 2016), 21. DOI: http://dx.doi.org/10.3390/bioengineering3040021

[22] Tania Pereira, Kais Gadhoumi, Mitchell Ma, Xiuyun Liu, Ran Xiao, Rene A. Colorado, Kevin J. Keenan, Karl Meisel, and Xiao Hu. 2020. A Supervised

Approach to Robust Photoplethysmography Quality Assessment. *IEEE Journal of Biomedical and Health Informatics* 24, 3 (2020), 649–657. DOI: http://dx.doi.org/10.1109/JBHI.2019.2909065

[23] Aysan Mahmoudzadeh, Iman Azimi, Amir M. Rahmani, and Pasi Liljeberg. 2021. Lightweight Photoplethysmography Quality Assessment for Real-time IoT-based Health Monitoring using Unsupervised Anomaly Detection. *Procedia Computer Science* 184 (2021), 140–147. DOI: http://dx.doi.org/https://doi.org/10.1016/j.procs.2021.03.025

[24] Steve Klabnik and Carol Nichols. 2018. *The Rust Programming Language*. No Starch Press, USA.

[25] GitHub/rust-lang. 2024. rust. (April 2024). https://github.com/rust-lang/rust

[26] GitHub/rust-lang. 2024. crates.io | The Rust community's crate registry. (07 2024). https://crates.io

[27] GitHub/deviceplug. 2024. btleplug. (07 2024). https://github.com/deviceplug/btleplug

[28] GitHub/rust-ndarray. 2024. ndarray. (07 2024). https://github.com/rust-ndarray/ndarray

[29] GitHub/tokio-rs. 2024. tokio. (07 2024). https://github.com/tokio-rs/tokio

[30] Mozilla. 2024. The UniFFI user guide. (07 2024). https://mozilla.github.io/uniffi-rs/

[31] Apple. 2024. SwiftUI | Apple Developer. (07 2024). https://developer.apple.com/documentation/swiftui/

[32] Apple. 2024. SwiftCharts | Apple Developer. (07 2024). https://developer.apple.com/documentation/charts

APPENDIX A: CORE API DOCUMENTATION

The Core API of VitalVision consists of several structures and methods and provides a structured interface for frontends to interact with. This appendix serves as a comprehensive reference for researchers and developers who may use or extend the VitalVision system. The following sections detail the key components of this API, starting with the structures and progressing to the main interface.

Device Structure

The `Device` structure represents a single wearable device, encapsulating all relevant information about its state and capabilities:

```
pub struct Device {
    id: String,
    // Unique serial identifier written on the
    // wearable PCB and labeled on the casing
    serial: u16,
    // Human readable name of the device
    name: String,
    // Battery level as a percentage (0-100)
    battery: u8,
    // Clock drift in microseconds relative to
    // the internal clock after initial setup
    drift_us: i64,
    // Connection status; disconnected
    // devices retain their last known
    // values
    connected: bool,
    // Data channels provided by this device
    channels: Vec<Channel>,
}
```

Channel Structure

The `Channel` structure defines an individual data source from a device, providing metadata about the type and quality of data it produces. It does however *not* contain the data itself (see the `newData()` callblack for that):

```
pub struct Channel {
    id: String,
    name: String,
    // can be CNT (Counter), ECG, or PPG
    channel_type: ChannelType,
    // Signal quality metric (0.0 to 1.0,
    // where 1.0 is optimal); None if
    // quality assessment is unsupported
    //or not yet performed
    signal_quality: Option<f32>,
}
```

Each channel is characterized by its identifier, name, type, and an optional signal quality metric. The channel type determines the nature and range of the data provided. Current wearables typically have at least a CNT channel, which represents a packet counter that loops through 0-255, and three PPG channels (green, red, and infrared), each with a signal range of 0-65 535. Devices connected with ECG leads also provide an ECG channel with a signal range of -32 768 to 32 767.

The signal quality field represents the analyzed integrity of the data, with 1.0 indicating optimal quality and lower values suggesting potential signal issues.

Core Delegate Trait

The preceding structures are maintained internally by the core and communicated to the frontend when changes occur. For this communication, a delegate pattern is employed. The `VVCoreDelegate` trait is to be implemented by the frontend and passed to the core upon instantiation:

```
pub trait VVCoreDelegate: Send + Sync {
    fn devices_changed(&self,
        devices: Vec<Device>);

    fn new_data(&self,
        uuid: String,
        data: Vec<Option<i32>>);
}
```

This trait defines the following two methods for core-to-frontend communication:

- `devices_changed`: This method is invoked when there's a change in device status or when new analysis results become available. It provides an updated list of all devices, allowing the frontend to refresh its display or trigger appropriate actions based on device changes.
- `new_data`: This method is called every time new datapoints are received for a channel. It receives the unique identifier of the device and a vector of optional integer values. The vector always contains `hist_size_api` elements, representing the latest data points. If fewer than `hist_size_api` points are available, the vector is padded with `None` values.

Note: Since the delegate is called inline in the data processing queue, implementers of these methods should ensure minimal processing time, particularly for `new_data`, to prevent bottlenecks in the data processing.

Core Configuration

The `VVCoreConfig` structure allows for fine-tuned parametric adjustment of the core's behavior:

```
pub struct VVCoreConfig {
    // Number of data points provided in the
    // delegate callback
    pub hist_size_api: u32,
    // Number of data points considered for
    // data analysis
    pub hist_size_analytics: u32,
    // Number of new points required before
    // running a new analysis
    pub analysis_interval_points: u32,
    // Maximum allowable initial clock drift
    pub max_initial_drift_ms: u32,
    // Interval in seconds for automatic
    // clock resynchronization
    pub sync_interval_sec: u64,
```

```
    // Flag to enable virtual devices and
    // signals for development and testing
    pub enable_mock_devices: bool,
    // Parameters for ECG signal analysis
    pub ecg_analysis_params:
        ECGAnalysisParameters,
    // Parameters for PPG signal analysis
    pub ppg_analysis_params:
        PPGAnalysisParameters,
}
```

Core Interface

The `VVCore` interface serves as the primary facade through which frontends interact with the VitalVision core library. This facade simplifies the interactions with the underlying subsystems by providing a higher-level, simplified API for initialization and control:

```
impl VVCore {
    // Initialize with config and delegate
    pub fn new(
        config: VVCoreConfig,
        delegate: Arc<dyn VVCoreDelegate>
        ) -> Self
    // Start device discovery and connection
    pub fn start_ble_loop(&self)
    // Manually synchronize device clocks
    pub fn sync_time(&self)
    // Disconnect all devices to save battery
    pub fn pause(&self)
    // Reconnect the devices
    pub fn resume(&self)
}
```