Title

RT-TRAQ: An algorithm for real-time tracking of faint quasi-periodic signals in noisy time series

Authors

Joarder, Rishad

Kasap, Begum

Ghiasi, Soheil

Publication Date

2023-06-01

DOI

10.1016/j.smhl.2023.100392

Peer reviewed

HHS Public Access

Author manuscript

Smart Health (Amst). Author manuscript; available in PMC 2023 November 15.

Published in final edited form as:

Smart Health (Amst). 2023 June ; 28: . doi:10.1016/j.smhl.2023.100392.

RT-TRAQ: An algorithm for real-time tracking of faint quasi-periodic signals in noisy time series

Rishad Joarder[a,*], **Begum Kasap**[a], **Soheil Ghiasi**[a]

[a]Dept. of Electrical and Computer Engineering, University of California Davis, Davis, CA 95616, USA

Abstract

We present an algorithm for live tracking of quasi-periodic faint signals in non-stationary, noisy, and phase-desynchronized time series measurements that commonly arise in embedded applications, such as wearable health monitoring. The first step of Rt-Traq is to continuously select fixed-length windows based on the rise or fall of data values in the stream. Subsequently, Rt-Traq calculates an averaged representative window, and its spectrum, whose frequency peaks reveal the underlying quasi-periodic signals. As each new data sample comes in, Rt-Traq incrementally updates the spectrum, to continuously track the signals through time. We develop several alternate implementations of the proposed algorithm. We evaluate their performance in tracking maternal and fetal heart rate using non-invasive photoplethysmography (PPG) data collected by a wearable device from animal experiments as well as a number of pregnant women who participated in our study. Our empirical results demonstrate improvements compared to competing approaches. We also analyze the memory requirement and complexity trade-offs between the implementations, which impact their demand on platform resources for real-time operation.

Keywords

quasi-periodicity; quasi-periodic wave; real-time monitoring; noise

1. Introduction

Many natural systems display irregular periodicity stemming from the internal mechanisms that drive them. The examples range from physiological systems like cardiovascular motion Sharma (2009), glucose-insulin regulatory system Rajagopal et al. (2020), cancer cells Cabello et al. (2019) to climate systems Snyder et al. (2011) like glacial-interglacial cycles. Analysis of irregular periodicity plays a major role in fields like motor fault detection Bonello & Pham (2012), predicting economic conditions, text encryption Tamba et al. (2019), etc. Systems like these are often affected by many internal and external disturbances, which force them to abruptly reset. This causes phase jumps within the periodic nature of these signals, often termed as quasi-periodicity. Nature is full of these noisy quasi-periodic signals.

*Corresponding author: rrjoarder@ucdavis.edu;.

Quasi-periodic signals are very hard to analyze using regular frequency domain methods. This is because they are made up of periodic waves with unrelated phases. Given a long enough time-series, each phase will have its own complement. Taking the Fourier Transform, the complementary components will cancel each other out. This essentially vanishes the quasi-periodic signal from the frequency domain. Fourier analysis of quasi-periodic waves requires special treatment.

There have been many studies focusing on quasi-periodic peak detection. They involve methods like phase rectification Bauer et al. (2006), wavelet transforms Gregoire et al. (2011), Hilbert transforms Benitez et al. (2001), K-means clustering Mehta et al. (2010), artificial neural network Pini et al. (2021), Markov models Coast et al. (1990), histogram Sezan (1990), time warping Makihara et al. (2011), fusing multiple data sources Kasap et al. (2021a) to stochastic resonance Deng et al. (2006).

There are many cases where quasi-periodic waves need to be monitored in real-time. Think of a nurse trying to place a probe on a patient's body. The probe placement will have an impact on the signal quality, which would ultimately affect medical decision-making. Having a live feedback while placing the probe makes the whole process more robust. There are many use cases where a real-time quasi-periodic wave tracing can bring a lot of utility. This is especially true for wearable health technologies. However, none of the methods mentioned above allow for such a real-time analysis of data. We propose an algorithm that aims to address this research gap.

In this paper, we propose a real-time algorithm for generating spectrum peaks corresponding to quasi-periodic waves in a low-SNR environment. We also propose several alternate implementations for this algorithm. We implement a very simple spectrum peak tracking algorithm on top of this to test it out on real data. We show the effectiveness of this approach by tracking maternal and fetal heart rate using non-invasive photoplethysmography (PPG) data collected by a wearable device. We use two different datasets, one collected from animal trials and another from pregnant women who participated in our study. We believe, this method can serve as a pre-processing step for tracking quasi-periodic waves in real-time.

2. Motivating Example

Transabdominal Fetal pulse Oximetry (TFO) uses light to non-invasively determine fetal oxygen saturation in a pregnant patient Zourabian et al. (2000); Fong et al. (2020b, 2021). It does so by continuously shining pulsating near-infrared light through the maternal abdomen in a process known as Continuous Wave Near-Infrared Spectroscopy (CWNIRS). The resulting photoplethysmography (PPG) signal picks up the subtle changes in blood volumes caused by maternal and fetal pulsations. A TFO system usually requires two wavelengths of light at specific frequencies for best performance Fong et al. (2017). Such a system is shown in Fig. 1. The TFO probe houses both pulsating sources and a set of detectors. The detectors closer to the source pick up light reflected from shallower layers—mostly from the maternal body. The farther detectors capture light bouncing off of deeper layers. Their PPG signal is a mixture of signals coming from both the maternal and the fetal body. The TFO system

can interpret fetal oxygen saturation (SpO_2) by analyzing the fetal component of the PPG signals Fong et al. (2020a); Kasap et al. (2021a).

The PPG signals can be thought of as high-frequency carrier signals created by the pulsating light sources, which are then amplitude modulated by a number of quasi-periodic physiological systems. These usually include maternal respiration rates (RR), maternal heart rates (MHR), fetal heart rates (FHR), Mayer waves, and occasionally patient motion. The optical signal obtained from these physiological processes is quite noisy owing to the distances involved. This is especially true for the fetal component. After demodulation, a PPG signal essentially reduces to a set of very noisy quasi-periodic waves.

Proper probe placement is absolutely crucial to obtain a usable PPG in a TFO system. Fetal SpO_2 determination methods are only effective as long as a strong fetal component is present in the PPG. To achieve this, the probe must be oriented in such a way that the optical path between the source & the detectors penetrates through the fetus. Owing to diverse body types and varied fetal orientations, proper placement of the TFO probe poses a strong challenge to physicians. A live monitoring system will increase robustness by eliminating the guesswork.

Moreover, maternal components of the PPG signal often interfere with the far weaker fetal component. This is where having multiple detectors come in handy. If placed correctly, the near-detector PPGs will consist of only the maternal components and almost no fetal parts. The near-detector PPG can then serve as a reference to denoise the far-detector PPG of any maternal interferences. Thus, proper probe placement also boosts signal separation algorithms.

The quasi-periodic FHR component of the PPG signal holds great importance in the TFO pipeline. Not only is the presence of a clean FHR peak a solid marker for a good probe placement, but the magnitude also directly factors into the fetal SaO_2 calculations. However, the FHR peaks often get buried beneath the noise floor due to the optical distances involved. It is difficult to decipher an FHR peak from a regular spectrogram. The spectrogram of a 90-second PPG signal crop, in Fig. 2, illustrates this problem. An online filtering algorithm that specializes in filtering noisy quasi-periodic signals is central to the performance of a real-time TFO system.

The ultimate goal of TFO is real-time patient monitoring. After the probe is placed, it will start producing a stream of noisy quasi-periodic PPG signals which need to be filtered and processed in real-time. The quality of the filtering algorithm directly affects the SpO_2 estimation. This processing needs to happen in real-time on the embedded system. Any offline algorithm working off of a data dump defeats the purpose of a TFO system.

3. Previous Work

Our work is a modification of the phase rectified signal averaging (PRSA) method Bauer et al. (2006). This section briefly explains the method and intuitions behind the original paper and its shortcomings in real-time application.

The PRSA algorithm uncovers quasi-periodic waves by averaging phase-synchronized windows from the original signal. The averaged window is termed as the PRSA wave. There are three steps to calculating a PRSA wave. *Step* 1: Pick a set of anchor points from the long time series. A multitude of strategies can be used for picking anchor points. For example, if a point's value is larger than one before it, that point can be chosen as an anchor. Another strategy could include picking points if the value is smaller. Both these strategies can be extended to using the mean value of M number of points before and after the point in question. *Step 2*: Crop windows of length (2×L+1) from the time series centered around each anchor point. We will call them PRSA windows. There will usually be quite a bit of overlap between the PRSA windows depending on the anchor points. The final PRSA wave will have the same length as each PRSA window. *Step 3*: Take a point-by-point average over all the PRSA windows. The average wave will then become the PRSA wave. The spectrum peaks of this PRSA wave reveal frequencies of the quasi-periodic waves.

A PRSA wave essentially summarizes the entire time series data into a single waveform. The intuition behind this can be understood by imagining the original signal as only a quasi-periodic sine wave. That is to say, the signal is composed of patches of phase-desynchronized noisy sine waves. Given that the noise does not overpower the sinusoid, the maximum variation is expected near zero phase. This is true on both ascending and descending sides of zero phase. In other words, all the anchor point-picking strategies mentioned above would choose the majority of the points near zero phases. Based on the exact strategy employed, these points would lie either on the ascending or the descending side. This makes most of the windows somewhat phase synced with respect to the underlying sine wave. Averaging the phase-synchronized windows cancels out the additive noise, leaving only the sine wave behind.

While this does work decently well in an offline setup, the definition for static PRSA does not lend itself to an online setup. Firstly, the anchor points are not directly defined to be reusable between segments of overlapping data. For a real-time implementation, this would mean recalculating the anchor points each time a new piece of data is added. Secondly, the anchor points at the edges of data segments are discarded in static PRSA. This is because the windows associated with those anchors will be partially empty. This cuts down on the number of windows available for averaging for the PRSA wave and degrades performance. If we want to do an online implementation with some sort of running window, this will put limits on the size of the window. Quite a few modifications are required to achieve a real-time version of the PRSA algorithm.

4. Rt-Traq Algorithm: Noise Reduction through Judicious Averaging

We go into the overview of the Rt-Traq algorithm, several alternate implementations, and their respective system resource requirements in this section. We also discuss a simple peak tracker that is stacked on top of our Rt-Traq implementations, which allows us to evaluate our method on TFO signals.

4.1. Core Components in Rt-Traq Wave Creation

4.1.1. Picking Anchor Points—The first step to generating an Rt-Traq wave is strategically picking anchor points from the time series data. In our implementation, we only consider the most recent data point as a new anchor point candidate. We pick the latest point as an anchor if the mean of the M most recent points is larger than the mean of the M points that appeared immediately before that. The M most recent points also include the anchor point candidate. In total, $2M$ points need to be compared each time a new sample comes in. For simplicity, we choose M as 1 for the rest of the paper (i.e., compare the most recent point with the second most recent point).

At the beginning of any data stream, the system will not have seen 2M data points to decide on an anchor. For this edge case, we assume the older data points as zeros.

4.1.2. Rt-Traq Windows—We define an Rt-Traq window as the array of an anchor point and the immediate $L - 1$ sample points appearing before it. This gives a total window length of L samples. Each anchor point is associated with an Rt-Traq window.

Judiciously choosing anchor points forces the corresponding windows to be in near phase-sync with one another. This property will hold as long as L is chosen to be larger than the time period of all underlying periodic/quasi-periodic waves with the time series. Averaging over all windows amplifies the periodic/quasi-periodic components.

4.1.3. Data Buffer—Data Buffer stores the L the most recent data point. If the current anchor point candidate is chosen, its corresponding Rt-Traq window is ready and waiting to be processed in the Data Buffer.

The Data Buffer array also makes the process of picking anchors more memory efficient, since all the relevant points necessary to make the decision are already stored within the buffer. This statement remains true as long as M is chosen such that $L \geq 2M$.

Data Buffer is implemented by using a ring buffer data structure. Each piece of new data from the stream overwrites the oldest piece of stored data.

4.1.4. Window Buffer—Once an anchor point is chosen, the array stored inside the Data Buffer gets passed on to the Window Buffer. Consequently, the Rt-Traq wave is calculated by taking a point-wise sum over all the windows stored in the Window Buffer. The Rt-Traq wave can be thought of as an average wave over the collection of the strategically picked, most recent windows from a long time-series.

In our implementations, each window is treated as a row and the Window Buffer as a stack of windows in a 2D matrix. Window Buffer's dimensions are $n_W \times L$, where n_W is the number of windows stored. The Rt-Traq wave is simply the column-wise sum over the Window Buffer of length L.

We propose three slightly alternate definitions for the Window Buffer, depending on which Rt-Traq windows get stored. We base our Rt-Traq implementation types based on these alternate definitions. These alternate styles are further discussed in the next section.

4.1.5. Generation Rate—We define the rate at which the external system requests an Rt-Traq wave as the generation rate. For a generation rate of m, we must provide Rt-Traq waves *once* every m samples. The generation rate is always predetermined in any system. It is different from the rate at which windows occur.

4.1.6. Putting It All Together—An overview of Rt-Traq wave creation is illustrated in Fig. 6. Each new piece of data from the stream is handed off to the Data Buffer. Depending on the previously stored data, we place a judgment on the current anchor point candidate. If the point gets chosen as an anchor, the Window Buffer is updated using the Data Buffer. The Window Buffer provides an Rt-Traq wave each time the external system requests one.

4.2. Variations in Rt-Traq Wave Implementations

We created three main types of Rt-Traq based on alternate definitions of the Window Buffer. Two of which are further divided onto Active and Lazy based on when the Rt-Traq wave is calculated, as shown in Fig. 4. The implementation details for each type of Rt-Traq are detailed in this section.

4.2.1. Constant Window Count Rt-Traq—In this variation, we always store a constant W number of L-length windows in the Window Buffer. As such, the Window Buffer always has a fixed size, shown in Fig. 5(a). Whenever a new window is added, the oldest one is discarded. Window Buffer is internally implemented as another ring buffer. In our implementation, each online window is stored as a row. The Window Buffer dimensions are $W \times L$. The Window Buffer is initialized as all zeros. The Rt-Traq wave is simply defined as the column-wise sum of the Window Buffer.

The Constant Window Count Rt-Traq wave can be computed both lazily or actively.

> **Active Implementation:** Rt-Traq wave is updated each time a new Rt-Traq window is available. This involves a point-by-point addition and subtraction of the latest and discarded window respectively with the previous Rt-Traq wave. The Rt-Traq wave is saved and is available for whenever it is requested.
>
> **Lazy Implementation:** Rt-Traq wave is calculated only when externally requested by taking a column-wise sum of the Window Buffer.

4.2.2. Constant Sample Count Rt-Traq—In this implementation, we only store the Rt-Traq windows which occurred within the most recent **n** samples in the Window Buffer. Any windows which occurred before that are removed from the Window Buffer during updates. The Window Buffer is implemented as a linked list, with each window being stored as an individual array within the linked list. This is illustrated in Fig. 5(b). We also save an extra element documenting the sample number for when each window occurred. The algorithm uses this information to discard windows that occurred more than **n** samples ago during each update. In our implementation, the latest window is appended at the end of the linked list. In other words, the first element on the list contains the oldest window. The length of the linked list is not fixed. Some parts of the signal might contain more windows than others. Some intervals of the signal might not even contain any windows whatsoever. In cases like these, the Rt-Traq wave is just defined as all zeroes. Situations like these occur at

the beginning of the data stream when there are not enough samples to get the first window. It might also happen during long stretches of monotonic rise/fall, when the anchor picker cannot choose any anchors. This implementation follows the definition of the original PRSA more closely.

Calculating the Constant Sample Count Rt-Traq wave is a 2-step process. In the first step, we update the linked list by discarding older windows. To do this, we traverse down the list and determine the last window which needs discarding using their timing information. Because of how the linked list is set up, any window preceding this on the list will also be discarded. The second step involves taking the column-wise sum of the Window Buffer to update the Rt-Traq wave. These two steps combined generate the new Window Buffer for the following updates and produce the current Rt-Traq wave. The Constant Sample Count Rt-Traq wave can be computed both lazily or actively.

Active Implementation: The active implementation updates its Rt-Traq wave with each new sample. During each update, the active implementation looks at the timing information only on the oldest window (In our case, the first). If it is old enough, the window is discarded. If the current Data Buffer gets chosen as a window, it is also added at the end of the Window Buffer. Using these two windows, The Rt-Traq wave is immediately updated and stored when the external system requests it.

Lazy Implementation: In the lazy implementation, both update steps are only calculated when an Rt-Traq wave is requested externally.

4.2.3. Decaying Average Rt-Traq—

Using a decaying average equation, the Rt-Traq wave is directly calculated and stored in the Window Buffer for this style of implementation. The Rt-Traq wave gets updated each time a new Rt-Traq window becomes available. The decay equation we used is given as,

$$RtTraq_{wave} = new\_window + \alpha \times RtTraq_{wave}$$

Where, $0 < \alpha < 1$ is the decay factor. A large decay factor fades out the older windows faster. The impact of the i^{th} most recent window is damped by a factor of α^{i-1}. This method is much more memory efficient compared to the rest. Decaying Average Rt-Traq can only be implemented actively, as the Rt-Traq wave needs to be updated with each new window. The update itself is a very simple weighted sum.

4.3. System Resource Requirements for Rt-Traq

We looked at three theoretical system resource requirements for each style of Rt-Traq implementation. For the theoretical calculations, we define a few key parameters. We assume that data is always sampled at f Hz. We are required to provide a new Rt-Traq wave externally at the generation rate, m. That is to say, the external system requests an Rt-Traq wave once every m samples or once every f/m seconds. Each time a new Rt-Traq wave is created internally, we call this an *update*. For the Lazy implementations, the update frequency and the generation rate are essentially the same things. For the active ones, however, updates occur fortuitously whenever a new window is available. This distinguishes

the update rate from the generation rate. We make the assumption that a single anchor point is expected to appear in every two samples. This leads to an expected anchor point rate of $f/2$ per second. The complexity for each individual update, memory requirements and expected update frequency are shown in Table 1 under these assumptions.

The lazy implementations have update complexity, but it boasts a lower update frequency. This is thanks to the **m** in the denominator in 1. For most practical use cases, we do not require a new Rt-Traq wave for every new sample. It is perfectly fine for the generation rate to be in the range of once every hundred or even a thousand samples. Such large values of **m** favor the lazy implementations over their active counterparts from a computational perspective.

From a memory perspective, Decaying Average Rt-Traq shows better performance. Since this implementation stores a single window, which also doubles as the Rt-Traq wave. This is the premier choice when onboard memory is limited.

4.4. Peak Tracking Add-on

We attach a simple spectrum peak-tracker to the output of our Rt-Traq wave generator to evaluate the effectiveness of our method on the dataset. The evaluation pipeline is shown in 6. With new data coming in, Rt-Traq algorithm creates a new wave at the generation rate (once every m samples). The peak-tracker locks on to a frequency peak and traces it throughout the Rt-Traq waves.

Initially, the spectrum needs to stabilize before we can start tracking peaks. At the beginning of any data stream, there is very little data for the Rt-Traq to generate a proper wave. The Rt-Traq spectrums created at this stage are often misleading. We wait L number of samples before locking on a peak.

Once stabilized, the frequency peak is updated with each new Rt-Traq wave. During each update, we constrict the search to a small band around the current peak. This is based on the assumption that the frequency peaks should not stray too far between samples. We also assume that the frequency peaks would shift smoothly. In other words, the derivatives should be continuous with respect to time. To ensure this, we assign twice as much weight to whichever direction the frequency peak is currently moving. So if a frequency peak is rising, the system assumes that it more like to rise or stay fixed in the next step. It places a lower bet on the peak falling. This allows the Rt-Traq to lock onto a single frequency peak and smoothly track through the data stream.

5. Experiments

5.1. Experimental Setup for Transabdominal Fetal Pulse Oximetry

We test out our algorithms on two separate PPG datasets collected from a handheld TFO system. The first one is obtained from 5 pregnant-sheep studies with multiple experiments. The procedures employed during animal studies followed a strict protocol approved by UC Davis Institutional Animal Care and Use Committee (IACUC) Vali et al. (2021). The second dataset is collected from pregnant women who volunteered for the study Kasap

et al. (2021b). The protocol used during the human patient studies is approved by UC Davis Institutional Review Board (IRB). This study has had 9 volunteers signing up so far, with some patients coming in for second visits and some never showing up. The TFO system contains a total of five detectors placed at varying distances from the near-infrared light sources, each collecting PPG data at 8kHz Fong et al. (2020b). The animal studies also include ground truth FHR & MHR values, which are recorded using hemodynamics. The human patient studies also include ground truth FHR & MHR via Doppler ultrasound transducers. For this paper, we chose the first two experiments of the last 3 sheep from the first study and every available patient from the second study. The total length of the downsampled data for each experiment is given in Table 2 and Table 3 for the animal and human experiments, respectively.

5.2. Performance Metric

The purpose of the experiments was to evaluate the performance of Rt-Traq in strengthening up the quasi-periodic waves beyond the noise floor in real-time. To achieve this, we apply Rt-Traq to the demodulated PPG data continuously and obtain a set of Rt-Traq waves. We attempt to lock in and track FHR & MHR using our peak detector through time. Comparing them against the known ground truth FHR (and MHR when available) allows us to judge the validity of our real-time spectrum cleaning technique.

We use mean absolute error (MAE) as our performance metric. The frequency peaks are calculated and compared against the ground truth every second. We calculate the error for each experiment individually, as well as an overall error rate for each dataset. We limit ourselves to only two implementations in this paper. The *Constant Window Count* and *Decaying Average*. MAE is used to compare the quality of each implementation.

For the sheep studies, we use both MHR & FHR. For the human patient studies, the MHR was unavailable for a few of the visits. Even for the visits where it was available, the MHR was missing for some parts of the experiment due to technical issues. So we ignored the ground truth MHRs for the human dataset and focused mainly on the FHR.

5.3. Generating Rt-Traq Data

We generated Rt-Traq data using only a single detector demodulated signal for simplicity. For the choice of detector, we ignored the three closest detectors, as they often times did not contain any FHR traces. The farthest detector was ignored for its low SNR. This leaves the fourth-closest detector as the best choice overall. For the modulation frequency, we picked the demodulated PPG signal for the 850nm wavelength source. This is because larger wavelengths theoretically provide better penetration depth. We did not attempt any multi-detector fusion techniques.

We fed offline data to our implementation one at a time to mimic a real-time setup with a generation rate of $m = 80$ (i.e., one Rt-Traq wave generated per 80 samples or per second). The generation rate is matched to the ground truth FHR & MHR sampling rates. This allows for a one-to-one comparison. This way, we were able to mimic a real-time scenario from offline data for evaluation purposes.

Some pre-processing steps were applied to the demodulated data for ease of evaluation. We downsampled the original 8kHz PPG signals to 80Hz for ease of computation. We also highpassed the signals at 0.1 Hz. We always normalize the spectrum magnitude to be between 0 and 1. These steps are not necessarily required, but it makes the results easier to visualize.

For visualization, we stitch the individual Rt-Traq wave spectrums vertically to create a spectrogram. If the Rt-Traq is able to uncover any quasi-periodic or periodic waves hidden within the time series, they appear as bright vertical streaks on the spectrogram. Usually the Respiration Rate (RR), MHR & FHR would appear as bright vertical streaks in the spectrogram. These spectrograms can be passed on to any peak tracker to trace the quasi-periodic waves.

5.4. Peak Tracking Specifications for TFO

We configured our peak tracking add-on to track FHR as well as MHR & RR for a TFO setup. This is because the Rt-Traq spectrums generated for this dataset contain harmonics for both RR and MHR. The RR harmonics, especially, are very strong. It can confuse both the MHR & FHR tracker. Usually, the FHR is pretty faint. Even the MHR harmonics can confuse the FHR tracker. This is where tracking all signals come in handy. The traces from RR & MHR can be used to determine the harmonics, which in turn are fed back into the trackers. These are used as the avoid ranges. The search range is set to 0.2Hz for all setups. This means the Rt-Traq only searches in a 0.4 Hz frequency band centered around the current guess. The avoid ranges are set as a 0.2 Hz frequency band centered around the RR & MHR harmonics. This gives the MHR & FHR tracker a better context.

5.5. Case Study: Fetal Heart Rate Tracking in Sheep

5.5.1. Tuning the Parameters—Before going into the performance, we need to determine the operational parameters best suited for this type of analysis. Parameter tuning is relatively simple in Rt-Traq since we only have to worry about two parameters.

- Number of averaged windows – known as **W**, **n** or a depending on the implementation.
- Window length **L** - which determines the size of the Rt-Traq wave and in turn the spectral resolution in all the implementations. A small L leads to lower frequency resolutions and difficulties in determining spectrum peaks. While a larger L provides an improved frequency resolution at the cost of time resolution as well as increased complexity & memory requirements.

The product of these two parameters determines the length of the time series data being averaged in each Rt-Traq wave. A smaller value results in a low-latency yet noisy system. Such a system would show a faster response to real-time change at the cost of being weaker at reinforcing the quasi-periodic components. A larger value creates a system that is much slower to respond to live events yet very strong at reinforcing the underlying quasi-periodic signals. For any given application, the designer should choose the minimum L that satisfies the frequency error constraints while providing minimal latency and clean spectrum peaks.

For evaluation, we set one parameter as a constant and sweep over the second one. We measure the MHR & FHR MAEs over each sheep round individually. These two errors serve as our metric for picking the optical parameters.

Parameter Tuning for Constant Window Count Rt-Traq: We first set $L = 1028$ and sweep over a range of values of W. As shown in Fig. 7, the lowest MAEs are obtained at $W = 30$. We then set $W = 30$ and sweep over a range of values for L. From Fig. 8, the best results are produced by L in the range of 1200 to 1500. For the rest of this paper, we assume $W = 30$ & $L = 1280$.

Parameter Tuning for Decaying Average Rt-Traq: We use the same values of L as obtained from Constant Window Count Rt-Traq and plot the MHR & FHR MAEs for different values of α in Fig. 9. The lowest errors occur for an α around 0.4. For the rest of the paper, we assume $\alpha = 0.4$.

5.5.2. Applying Rt-Traq to the Animal Dataset—

Rt-Traq algorithm generates a series of L-length Rt-Traq waves, with each wave showing strong peaks for the underlying quasi-periodic signals. The time domain and frequency domain of one such wave is shown in Fig. 11. This one is created using the constant window count Rt-Traq with optimal parameters for this problem. Unlike the regular spectrum in Fig. 2, the Rt-Traq spectrum actually shows large peaks corresponding to the FHR & MHR.

The difference in spectrum between the raw PPG signal and the Rt-Traq filtered wave is better visualized in the spectrogram plots in Fig. 12. The quasi-periodic frequency peaks uncovered by the algorithm in each Rt-Traq wave are easy to trace throughout time in the filtered spectrogram. Whereas for the raw signal, the FHR is pretty much absent and the MHR is only present for a fraction of the periods. These strong peaks allow the algorithms to follow these frequencies much more easily throughout the experiment.

Some examples of frequency tracking using Rt-Traq waves and the peak tracker are shown in Fig. 10. The plots also show the ground truth values for both FHR & MHR. The accuracy for each experiment, using each implementation, is illustrated in Table 4.

5.6. Case Study: Fetal Heart Rate Tracking in Human Patients

For the human dataset, we used a similar empirical approach to pick the best operational parameters for this dataset. The results for both methods are tabulated in 5.

6. Results & Discussion

We compare the effectiveness of our online algorithm with an existing offline algorithm described in Kasap et al. (2021b). The mean absolute errors for both methods are shown in Table 5 for different human patient experiments. Despite being real-time, our method performs relatively well in comparison. Even outperforming the offline method in some patients.

The existence of multiple quasi-periodic signals leads to poor performance in some datasets. The Rt-Traq spectrum was able to pick up RR & its harmonics, MHR & its 2nd harmonic,

and FHR. This sometimes caused our spectrum peak tracker to lock on to a different signal. This is especially true during patches when the original signal peak disappeared from the spectrogram. These multiple peaks can be seen in Fig. 12. Having a view of the entire dataset can allow algorithms to easily bypass these pitfalls. But being real-time, locking onto a wrong peak carried the error forward, leading to large overall errors.

Multiple frequency peaks crossing each other confuses the spectrum peak detection algorithm. We see this behavior in Animal C, Experiment 2. This is shown in Fig. 13. Here, close to the end of the signal, the FHR crosses paths with the second harmonic of MHR. This causes the Rt-Traq to start following the MHR harmonic instead of the FHR. For this case as well, having a larger scope of view would as opposed to making a real-time decision would lead to better results.

Both Rt-Traq implementations show very similar results despite having different computational and memory requirements. On one hand, we have the Lazy Constant Window Count Rt-Traq with a lower computational complexity but a higher memory requirement. On the other hand, we have the Decaying Average with a higher computational complexity but a lower memory requirement. This means we can choose whichever implementation fits the platform requirements better without having to worry about performance.

The Rt-Traq spectrum generation essentially depends on only two operational parameters. The window length and the duration of the crop used for calculation. The low number of parameters makes it much easier to apply Rt-Traq spectrum generation to a wide range of problems without much tuning.

We tested a few different anchor point-picking strategies. This included different values of M, swapping the increasing condition for a decreasing condition in picking anchors, and weighted means. They did not provide any significant improvements. Still, anchor point-picking strategies specialized for a given problem is something that could be explored.

Data fusion may provide more reliable results for datasets with information from multiple channels. For example, in the TFO dataset, FHR traces can sometimes be present third-closest TFO detector. FHR traces are always present in the farthest detector. The Rt-Traq wave for both channels and both wavelengths can be merged to get a better estimate.

In this paper, we show a few different Rt-Traq implementations with similar performances which can trace quasi-periodic waves in real-time data. A more sophisticated filtering approach can be applied on top of Rt-Traq spectrum to be passed downstream for specific applications. For a frequency finding problem, even the peak tracker described here can be modified and extended to meet specific scenarios.

7. Conclusion

In this paper, we propose Rt-Traq, an algorithm capable of extracting and following quasi-periodic waves from real-time data. We showed the effectiveness of the Rt-Traq spectrum in continuously discovering FHR & MHR peaks in both animal and human PPG data. We also give designers the choice between a few alternative approaches to make necessary trade-offs

between system memory and computational requirements. Our algorithm can be included as a part of any embedded system's data processing pipeline, that is used in the live monitoring of quasi-periodic signals.

Acknowledgments

The authors would like to gratefully acknowledge support from National Science Foundation (NSF) grants 1838939 and 1937158, and the Eunice Kennedy Shriver National Institute of Child Health and Human Development (NICHD) grant R21HD097467.

The opinions expressed in this paper are the author's own and do not represent the funding organizations.

References

Bauer A, Kantelhardt JW, Bunde A, Barthel P, Schneider R, Malik M, & Schmidt G (2006). Phase-rectified signal averaging detects quasi-periodicities in non-stationary data. Physica A: Statistical Mechanics and its Applications, 364, 423–434. doi:10.1016/j.physa.2005.08.080.

Benitez D, Gaydecki PA, Zaidi A, & Fitzpatrick AP (2001). The use of the Hilbert transform in ECG signal analysis. Computers in Biology and Medicine, 31, 399–406. doi:10.1016/S0010-4825(01)00009-9. [PubMed: 11535204]

Bonello P, & Pham HM (2012). An investigation into the dynamic response of a squeeze-film damped twin-shaft test rig. In 10th International Conference on Vibrations in Rotating Machinery (pp. 79–88). Woodhead Publishing. doi:10.1533/9780857094537.2.79.

Cabello M, Ge H, Aracil C, Moschou D, Estrela P, Manuel Quero J, S. I. Pascu, & P. R. F. Rocha (2019). Extracellular Electrophysiology in the Prostate Cancer Cell Model PC-3. Sensors, 19, 139. doi:10.3390/s19010139. [PubMed: 30609788]

Coast D, Stern R, Cano G, & Briller S (1990). An approach to cardiac arrhythmia analysis using hidden Markov models. IEEE Transactions on Biomedical Engineering, 37, 826–836. doi:10.1109/10.58593. [PubMed: 2227969]

Deng H, Xiang B, Liao X, & Xie S (2006). A linear modulation-based stochastic resonance algorithm applied to the detection of weak chromatographic peaks. Analytical and Bioanalytical Chemistry, 386, 2199–2205. doi:10.1007/s00216-006-0858-7. [PubMed: 17115146]

Fong D, Knoesen A, & Ghiasi S (2017). Transabdominal fetal pulse oximetry: The case of fetal signal optimization. In 2017 IEEE 19th International Conference on E-Health Networking, Applications and Services (Healthcom) (pp. 1–6). doi:10.1109/HealthCom.2017.8210799.

Fong DD, Vali K, & Ghiasi S (2020a). Contextually-aware Fetal Sensing in Transabdominal Fetal Pulse Oximetry. In 2020 ACM/IEEE 11th International Conference on Cyber-Physical Systems (ICCPS) (pp. 119–128). Sydney, Australia: IEEE. doi:10.1109/ICCPS48487.2020.00019.

Fong DD, Yamashiro KJ, Johnson MA, Vali K, Galganski LA, Pivetti CD, Farmer DL, Hedriana HL, & Ghiasi S (2020b). Validation of a Novel Transabdominal Fetal Oximeter in a Hypoxic Fetal Lamb Model. Reproductive Sciences, 27, 1960–1966. doi:10.1007/s43032-020-00215-5. [PubMed: 32542541]

Fong DD, Yamashiro KJ, Vali K, Galganski LA, Thies J, Moeinzadeh R, Pivetti C, Knoesen A, Srinivasan VJ, Hedriana HL, Farmer DL, Johnson MA, & Ghiasi S (2021). Design and In Vivo Evaluation of a Non-Invasive Transabdominal Fetal Pulse Oximeter. IEEE Transactions on Biomedical Engineering, 68, 256–266. doi:10.1109/TBME.2020.3000977. [PubMed: 32746021]

Gregoire JM, Dale D, & van Dover RB (2011). A wavelet transform algorithm for peak detection and application to powder x-ray diffraction data. Review of Scientific Instruments, 82, 015105. doi:10.1063/1.3505103. [PubMed: 21280856]

Kasap B, Vali K, Qian W, Chak WH, Vafi A, Saito N, & Ghiasi S (2021a). Multi-Detector Heart Rate Extraction Method for Transabdominal Fetal Pulse Oximetry*. In 2021 43rd Annual International Conference of the IEEE Engineering in Medicine Biology Society (EMBC) (pp. 1072–1075). doi:10.1109/EMBC46164.2021.9630946.

Kasap B, Vali K, Qian W, Hedriana HL, Wang A, Farmer DL, & Ghiasi S (2021b). Towards Noninvasive Accurate Detection of Intrapartum Fetal Hypoxic Distress. In 2021 IEEE 17th International Conference on Wearable and Implantable Body Sensor Networks (BSN) (pp. 1–4). doi:10.1109/BSN51625.2021.9507036.

Makihara Y, Trung NT, Nagahara H, Sagawa R, Mukaigawa Y, & Yagi Y (2011). Phase Registration of a Single Quasi-Periodic Signal Using Self Dynamic Time Warping. In Kimmel R, Klette R, & Sugimoto A (Eds.), Computer Vision – ACCV 2010 (pp. 667–678). Berlin, Heidelberg: Springer. doi:10.1007/978-3-642-19318-7_52.

Mehta SS, Shete DA, Lingayat NS, & Chouhan VS (2010). K-means algorithm for the detection and delineation of QRS-complexes in Electrocardiogram. IRBM, 31, 48–54. doi:10.1016/j.irbm.2009.10.001.

Pini N, Lucchini M, Esposito G, Tagliaferri S, Campanile M, Magenes G, & Signorini MG (2021). A Machine Learning Approach to Monitor the Emergence of Late Intrauterine Growth Restriction. Frontiers in Artificial Intelligence, 4.

Rajagopal K, Bayani A, Jafari S, Karthikeyan A, & Hussain I (2020). Chaotic dynamics of a fractional order glucose-insulin regulatory system. Frontiers of Information Technology & Electronic Engineering, 21, 1108–1118. doi:10.1631/FITEE.1900104.

Sezan MI (1990). A peak detection algorithm and its application to histogram-based image data reduction. Computer Vision, Graphics, and Image Processing, 49, 36–51. doi:10.1016/0734-189X(90)90161-N.

Sharma V (2009). Deterministic chaos and fractal complexity in the dynamics of cardiovascular behavior: Perspectives on a new frontier. The Open Cardiovascular Medicine Journal, 3, 110–123. doi:10.2174/1874192400903010110. [PubMed: 19812706]

Snyder CW, Mastrandrea MD, & Schneider SH (2011). The Complex Dyanmics of the Climate System: Constraints on our Knowledge, Policy Implications and the Necessity of Systems Thinking. In Hooker C (Ed.), Philosophy of Complex Systems (pp. 467–505). Amsterdam: North-Holland volume 10 of Handbook of the Philosophy of Science. doi:10.1016/B978-0-444-52076-0.50017-1.

Tamba VK, Kengne R, Kingni ST, & Fotsin HB (2019). Chapter 14 - A Four-Dimensional Chaotic System With One or Without Equilibrium Points: Dynamical Analysis and Its Application to Text Encryption. In Boubaker O, & Jafari S (Eds.), Recent Advances in Chaotic Systems and Synchronization Emerging Methodologies and Applications in Modelling (pp. 277–300). Academic Press. doi:10.1016/B978-0-12-815838-8.00014-5.

Vali K, Kasap B, Qian W, Theodorou CM, Lihe T, Fong DD, Pivetti CD, Kulubya ES, Yamashiro KJ, Wang A, Johnson MA, Hedriana HL, Farmer DL, & Ghiasi S (2021). 974 Non-invasive transabdominal assessment of In-Utero fetal oxygen saturation in a hypoxic lamb model. American Journal of Obstetrics & Gynecology, 224, S604. doi:10.1016/j.ajog.2020.12.999.

Zourabian A, Siegel A, Chance B, Ramanujam N, Rode M, & Boas DA (2000). Trans-abdominal monitoring of fetal arterial blood oxygenation using pulse oximetry. Journal of Biomedical Optics, 5, 391. doi:10.1117/1.1289359. [PubMed: 11092427]

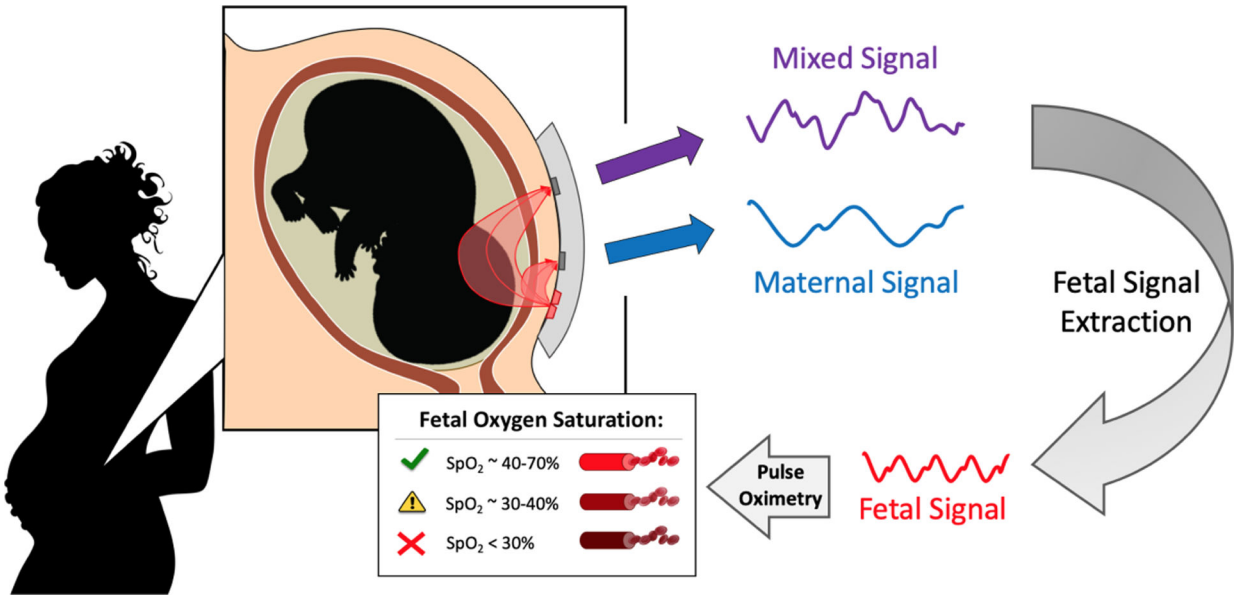


Fig. 1:
A High-level overview of Transabdominal Fetal Pulse Oximetry system Fong et al. (2020a)

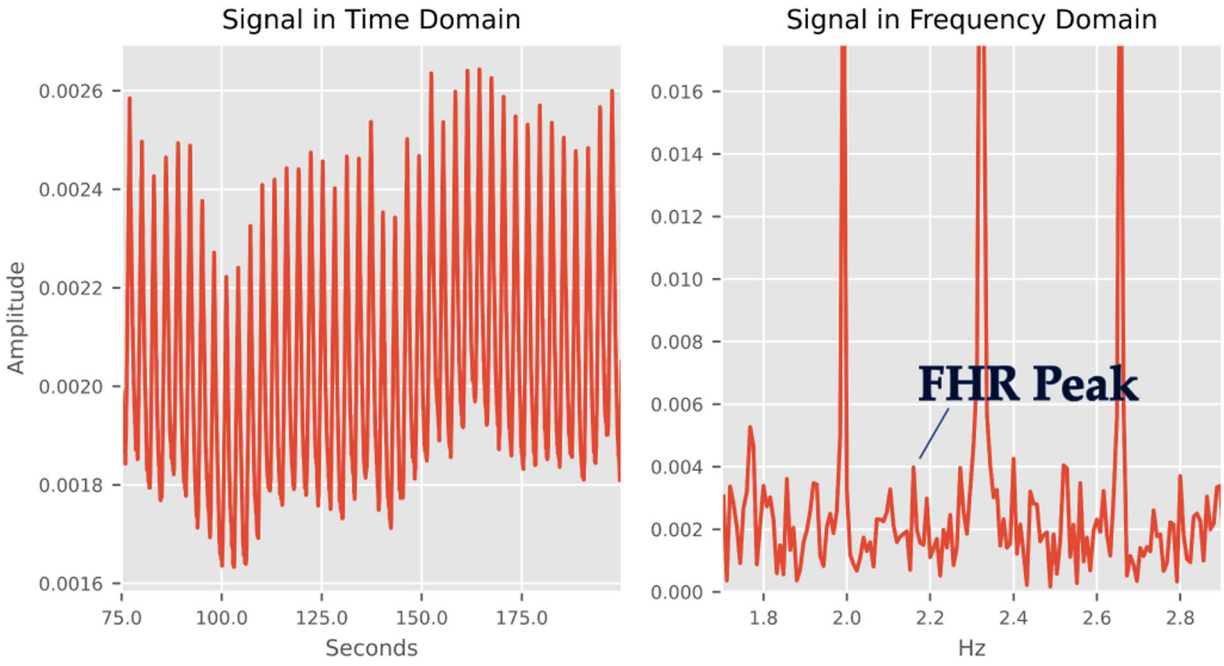


Fig. 2:

Time series plot & spectrum of a 90-second crop of demodulated PPG data (Animal A, Experiment 1). The ground truth Fetal Heart Rate (FHR) is provided by a hemodynamic monitor. Without any spectrum clean-up techniques, the FHR peak is very faint and surrounded by noisy peaks in the PPG spectrum. Isolating and monitoring the FHR peak from this spectrum in real-time poses a challenge

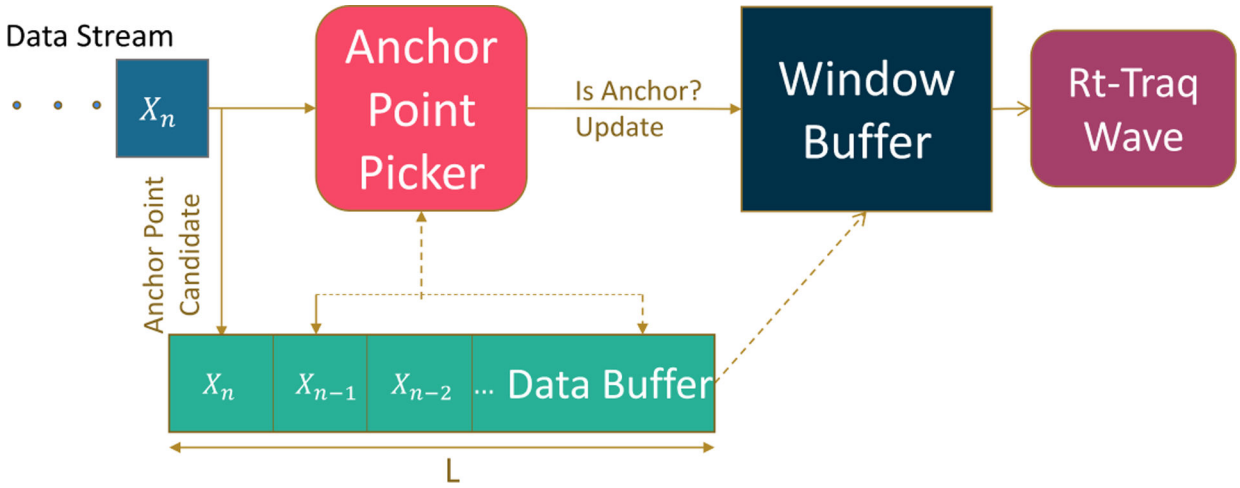


Fig. 3:

The core components of Rt-Traq Wave generation from a real-time data stream. Incoming data is fed to a ring buffer called the Data Buffer. The latest data point becomes the anchor point candidate and its corresponding window is saved in the Data Buffer. An anchor point-picking strategy gives the verdict using the Data Buffer. If chosen as an anchor point, the current window is stored in a secondary buffer, the Window Buffer. Rt-Traq wave is generated by averaging all the windows in the Window Buffer

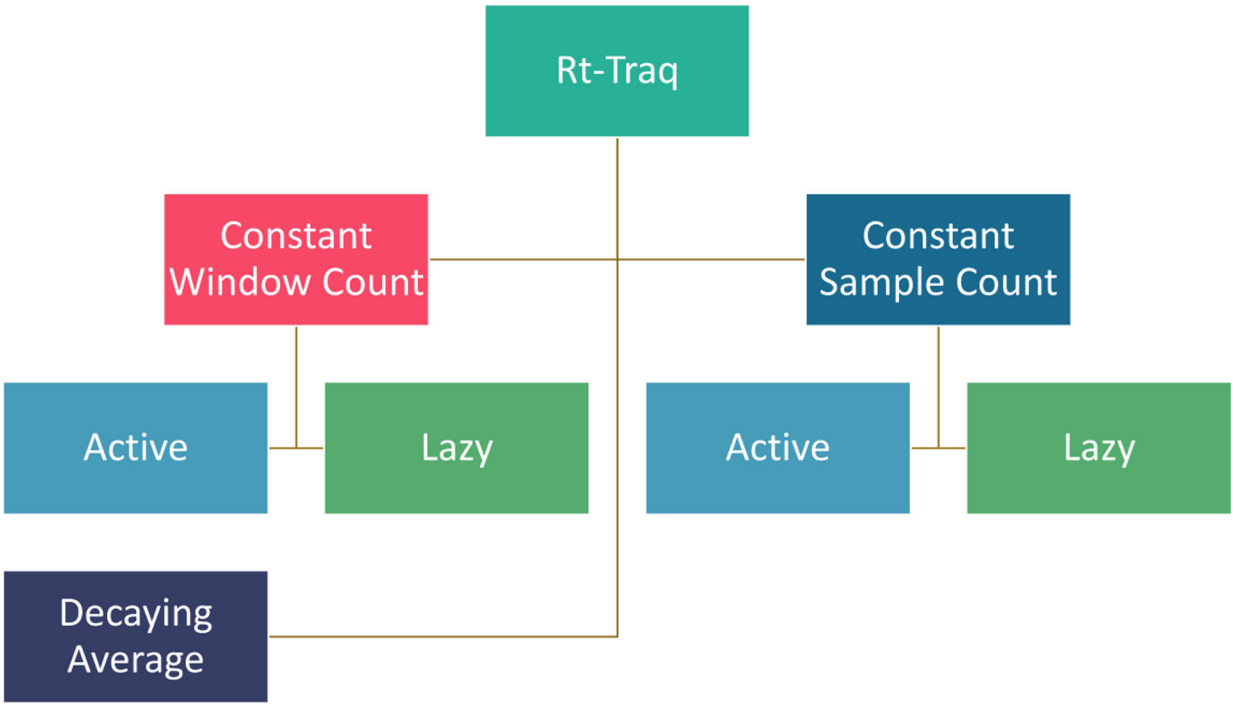


Fig. 4:
Styles of Rt-Traq Implementations

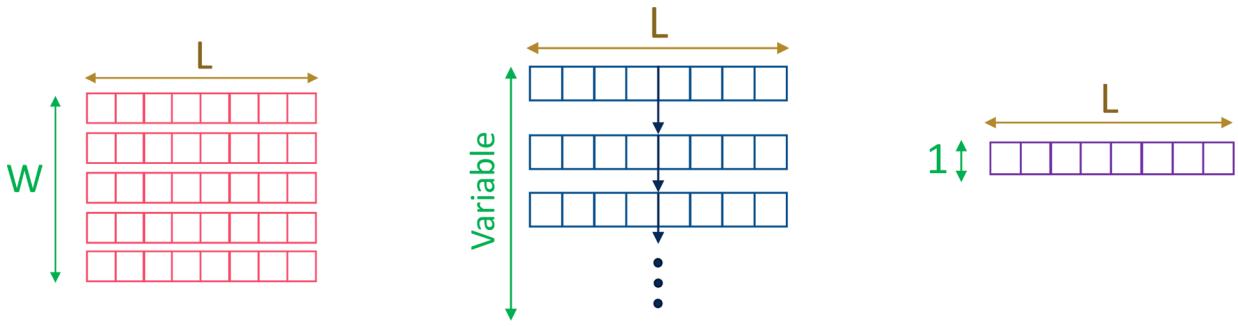


Fig. 5:

Three alternate definitions of the Window Buffer. In Constant Window Count Rt-Traq, the Window Buffer holds a constant W number of windows and is implemented as $W \times L$ fixed size 2D matrix (a). The Window Buffer holds any windows appearing in the n-most recent data samples in Constant Sample Count Rt-Traq. Implemented as a linked list of L-length arrays(b). In Decaying Average Rt-Traq, the Window Buffer stores a decayed average of all past windows in an L-length 1D array, which directly acts as the Rt-Traq wave(c).

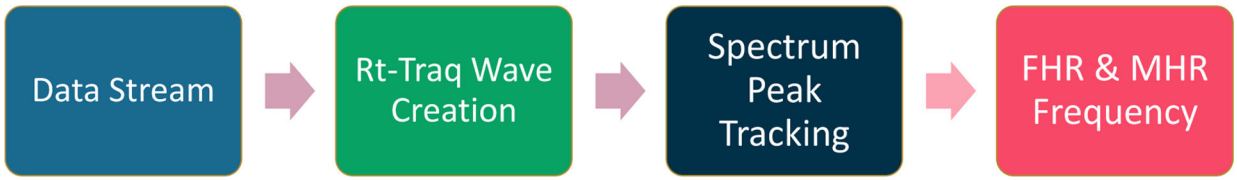


Fig. 6:
Overview of the Rt-Traq system used in evaluation

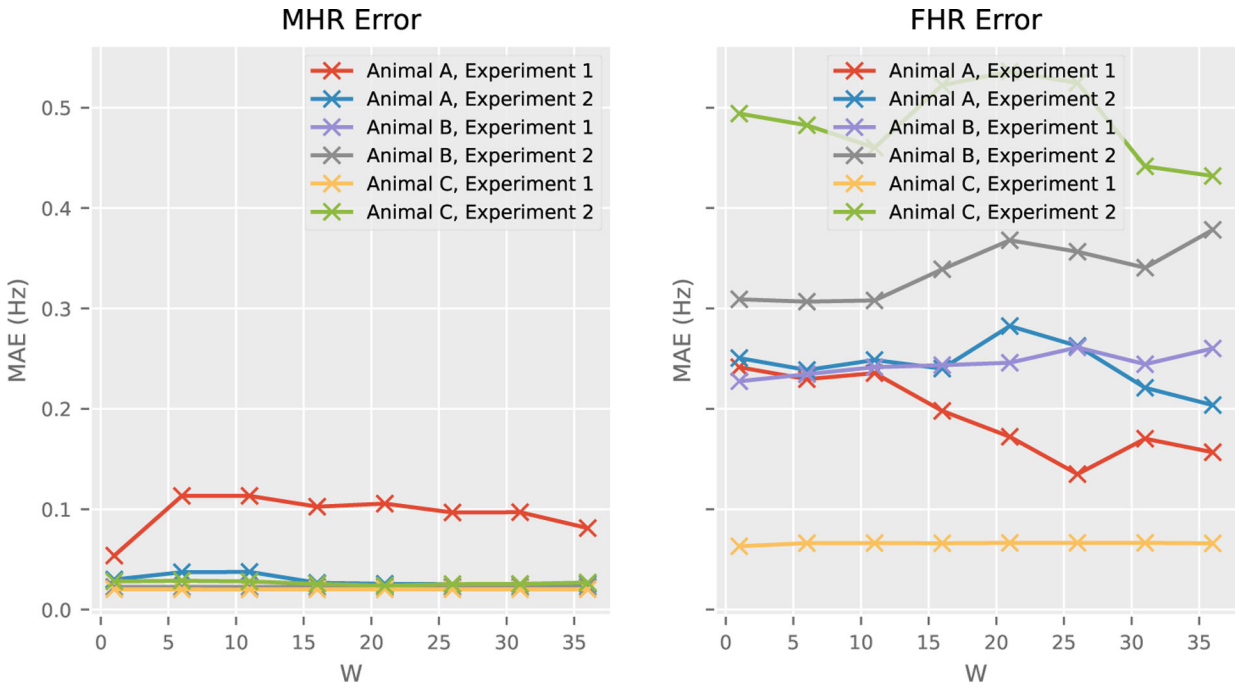


Fig. 7:

Effects of the parameter W on FHR & MHR MAEs for a fixed L=1024 in Constant Window Count Rt-Traq

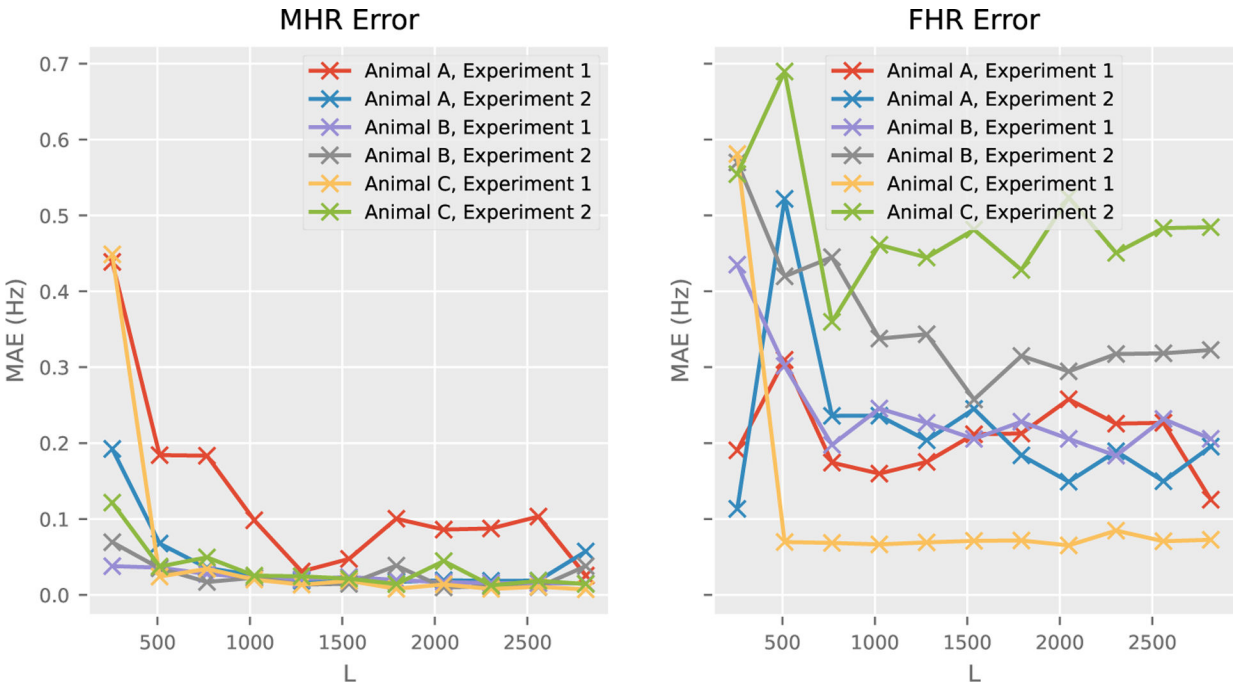


Fig. 8:

Effects of the parameter L on FHR & MHR MAEs for a fixed W=30 in Constant Window Count Rt-Traq

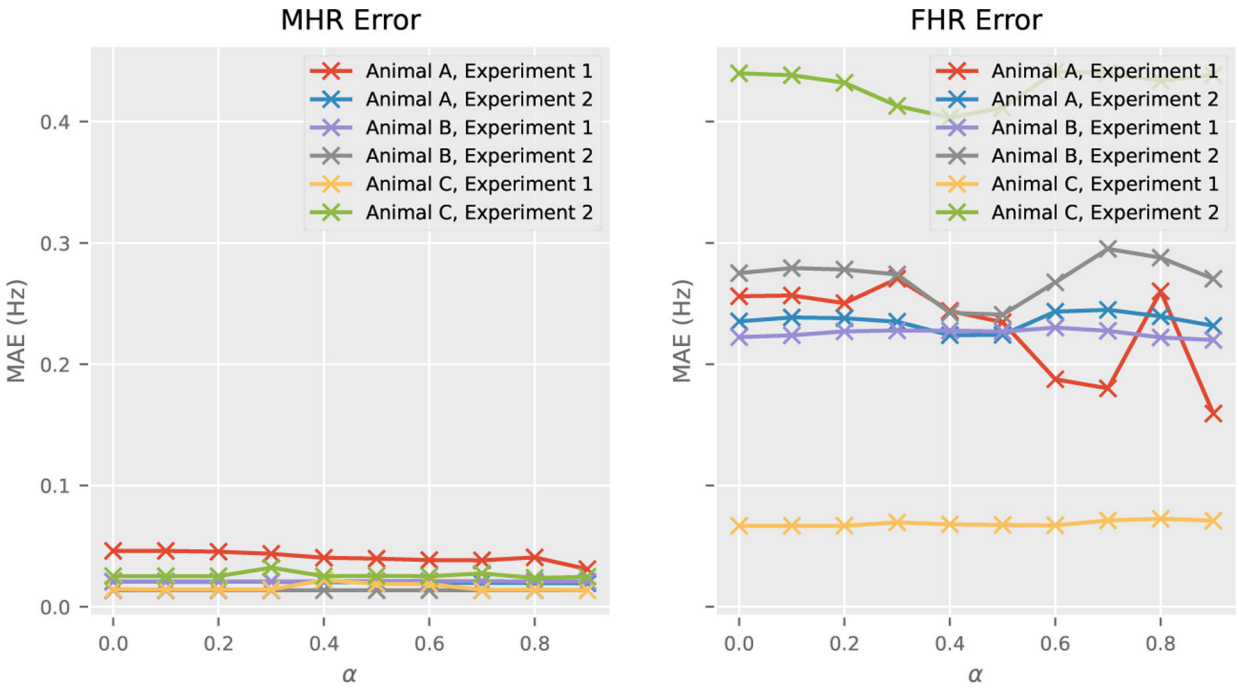


Fig. 9:

Effects of the parameter α on FHR & MHR MAEs for a fixed L=1080 in Decaying Average Rt-Traq

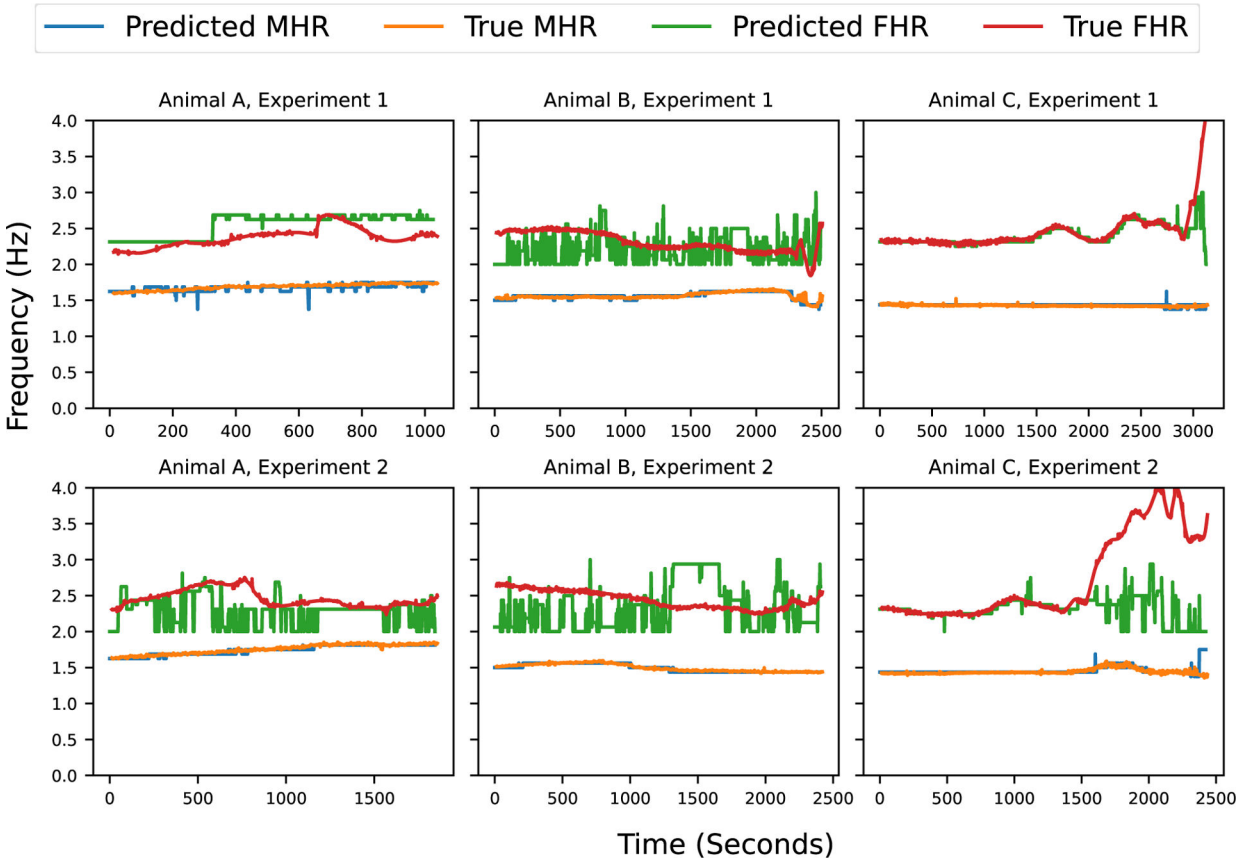


Fig. 10:

Some examples of MHR & FHR tracking on the waves generated via Constant Window Count Rt-Traq for animal and human patient studies, showing the effectiveness of our spectrum cleanup technique.

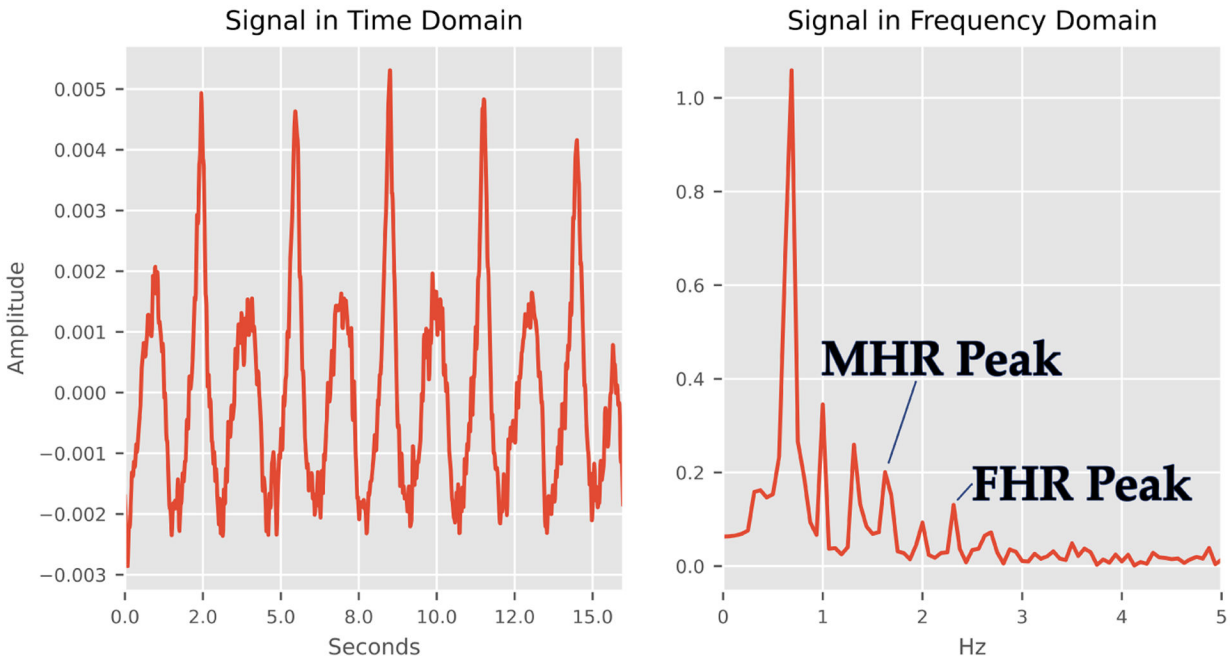
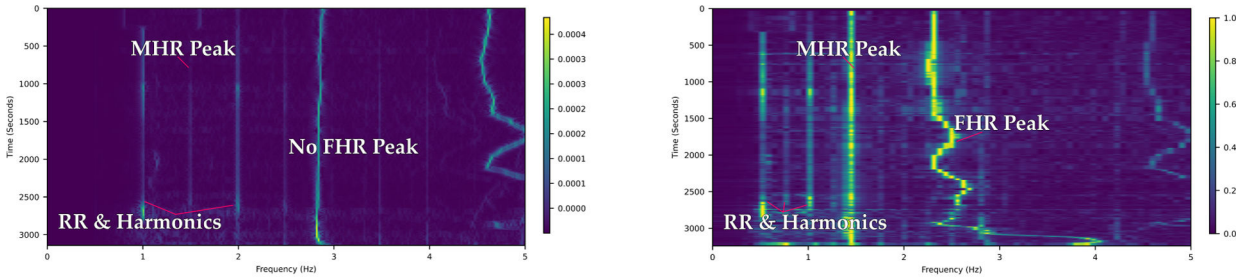


Fig. 11:

An Rt-Traq wave generated using Constant Window Count, which is able to reinforce and clearly show the FHR & MHR peaks in its spectrum. A series of Rt-Traq waves generated per second allows us to trace the frequency peaks over time. This Rt-Traq wave was generated with the optimal parameters, $W = 30$ & $L = 1280$.



(a) Regular Spectrum (b) Rt-Traq Spectrum

Fig. 12:

(a) Raw signal Spectrogram using a 50% overlap Tukey window vs. (b) Constant Window Count Rt-Traq Spectrogram created with W = 30 and L = 1280 for animal A, experiment 1. The FHR peaks are non-existent on the original spectrogram. The MHR is faintly visible but vanishes for a period of time. The Rt-Traq spectrum brings out the MHR & FHR very cleanly.

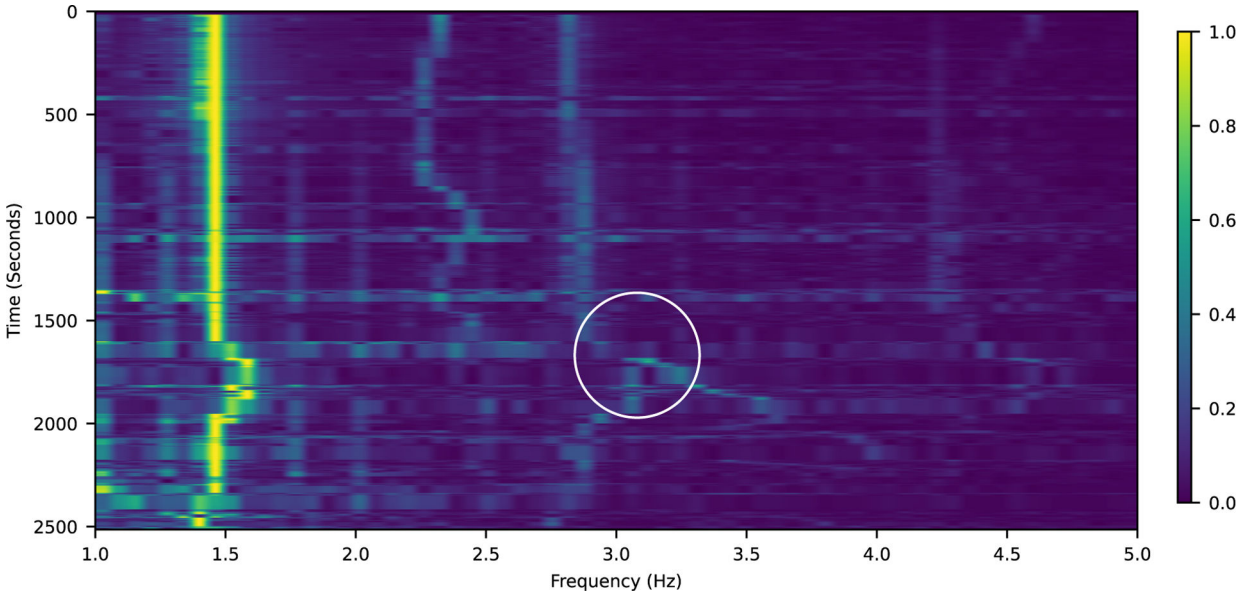


Fig. 13:

The peaks of FHR & MHR's 2nd harmonic crossing over each other at a certain point in time and confusing the peak tracker. This causes the tracker to follow the wrong peak after the crossover

Table 1:

System Resource Requirements of Rt-Traq Wave Generation

	Update Complexity		Memory		Expected Updates/Second	
	Lazy	**Active**	**Lazy**	**Active**	**Lazy**	**Active**
Constant Window Count	$O(W \times L)$	$O(L)$	$(W + 1) \times L$	$(W + 1) \times L$	f/m	$f/2$
Constant Sample Count	$O(n \times L)$	$O(L)$ / $O(1)$ (worst/best)	$(n + 1) \times L$	$\max(n, m)$ (worst)	f/m	f
Decaying Average	$O(L)$		L		$f/2$	

Table 2:

Dataset length for each of the animal experiments (after downsampling to 80Hz)

Experiment	Data Length
Animal A Experiment 1	82160
Animal A Experiment 2	147600
Animal B Experiment 1	199520
Animal B Experiment 2	192640
Animal C Experiment 1	250000
Animal C Experiment 2	193920

Table 3:

Dataset length for each of the human patient experiments (after downsampling to 80Hz)

Patient	Data Length
P1 V1	92240
P1 V2	82400
P3 V1	128480
P4 V1	100960
P5 V1	126880
P7 V1	113280
P8 V1	95520
P9 V1	68960
P9 V2	76160

Table 4:

Performance evaluation using MAE(in bpm) for the three types of Rt-Traq on the animal dataset

	Constant Window Count		Decaying Average	
	MHR	**FHR**	**MHR**	**FHR**
Animal A Experiment 1	1.8	12	2.4	14.4
Animal A Experiment 2	1.8	10.8	1.2	13.2
Animal B Experiment 1	1.2	13.8	1.2	13.8
Animal B Experiment 2	1.2	20.4	0.6	14.4
Animal C Experiment 1	0.6	4.2	1.2	3.6
Animal C Experiment 2	0.6	26.4	1.2	24.0
Per Experiment Average	1.2	14.4	1.2	13.8

Table 5:

FHR prediction error comparison between our real-time method with an existing offline method for the Transabdominal Fetal pulse Oximetry human patient study

	Existing Offline Method(MAE in bpm)	Our Online Method(MAE in bpm)	
		Constant Window Count	Decaying Average
P1 V1	20.4	13.2	15
P1 V2	6.0	26.4	23.4
P2 V1	5.4	11.4	18.6
P3 V1	13.8	17.4	18.6
P4 V1	4.8	14.4	18.6
P5 V1	10.2	22.2	19.2
P7 V1	5.4	16.2	13.2
P8 V1	9.6	30.6	27.6
P9 V1	12.0	11.4	18.0