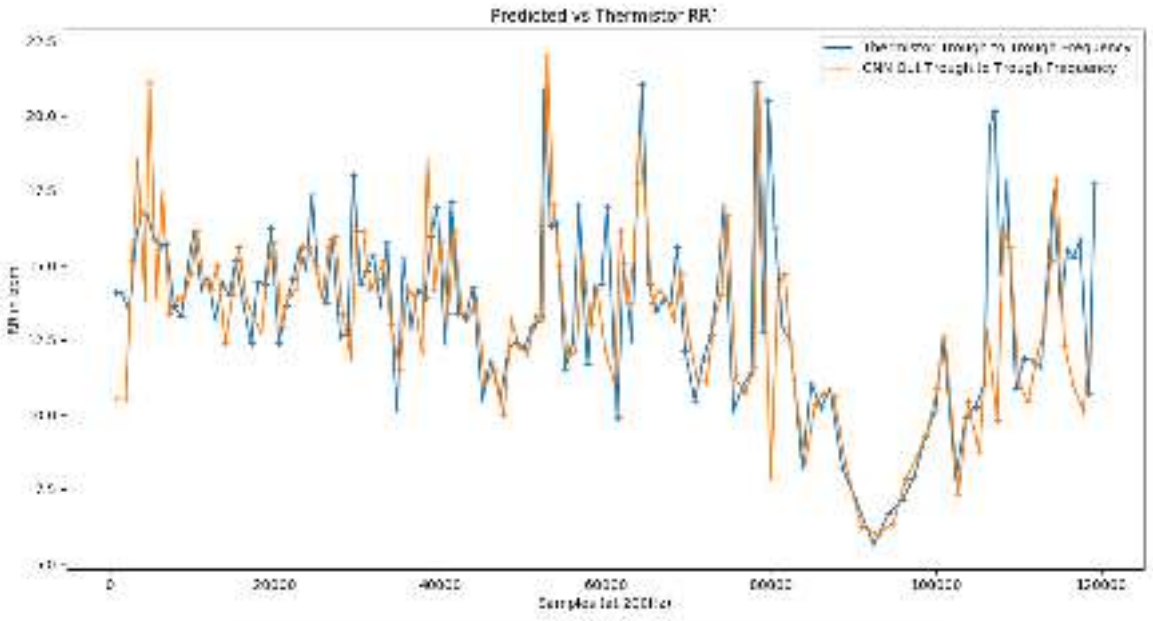
AN ALGORITHM TO INFER BREATHING RATE FROM PULSE OXIMETRY DATA

Curtis Huebner
Anmol Jawandha
Cheng Xie



Project Sponsors:
Sampath Satti
Prash Pandey

Project Number 1807
Applied Sciences 459
Engineering Physics
The University of British Columbia
2017-10-16

Executive Summary

Recently, there has been a spike in drug related deaths in the Vancouver area starting in 2016, with 938 deaths occurring in 2016. Among those deaths, a significant fraction can be attributed to opioid overdoses. The Vital Signs team aims to use a pulse oximeter, which is a non-invasive devices that provides photoplethysmography (PPG) signal to infer breathing rate for the purpose of detecting opioid overdoses. The main objective of this project is to facilitate the eventual development of an algorithm to estimating breathing rate from PPG signals.

In order to meet the project objectives, we developed an algorithm that uses a Neural Network to predict thermistor values using 4 input features. These are the amplitude of the PPG, two measures of the heart rate derived from the PPG and a filtered version of the PPG itself.The signal from the neural network is then post processed to yield an estimate of breathing rate. In addition, we provide a framework that makes it easy for researchers to add their own algorithms and obtain results on their own PPG data.

We collected our own dataset using a PPG sensor and a thermistor which provides truth labels for training and validation of our predictions. We find that the algorithm does well in capturing the overall breathing rate and how it changes over time. Importantly, our algorithm is able to capture the breathing rate well at lower frequencies, which signals the usefulness of our algorithm in the case of an opioid overdose. We report a root mean square error (RMSE) of +/- 2.5 - 3.5 breaths per minute (bpm) for instantaneous respiratory rate and +/- 1.1 - 2.1 bpm for STFT centroid respiratory rate (error in frequency space computed with a short time fourier transform) which better captures overall breathing rate over time.

There are several limits to our methodology. Most notably, all of the data that we collected came from 5 college age males. As such, we recommend that additional data be taken before making any strong claims about the performance of the methods developed. In addition, we also recommend that improvements be made to the PPG and thermistor data collectors. Finally, if the Vital Signs team decides to move forward with clinical opioid overdose testing, we recommend that additional methods of detecting opioid overdoses be tested in addition to PPG based respiratory rate estimation.

Table of Contents

List of Figures

List of Tables

Introduction

Recently, there has been a spike in drug related deaths in the Vancouver area starting in 2016, with 938 deaths occurring in 2016. Among those deaths, a significant fraction can be attributed to opioid overdoses. The Hatching Vital Signs team aims to address this problem of opioid overdoses by developing a cheap non-invasive sensor for detecting opioid overdoses as they occur.

Decline in respiratory rate is a good proxy for the occurence of an opioid overdose. There is large body of literature that details ways in which breathing rate can be inferred from a device know as a photoplethysmogram (PPG), along with a commercial PPGs which are capable of inferring breathing rate.

The main objective of this project is to facilitate the eventual development of an algorithm to detect/predict opioid overdoses using hardware developed by the vitalsigns team. As such the project aims to achieve the following objectives**:**

- Collect PPG signals from the ‘Max’ PPG sensor along with corresponding breathing rate signals and design algorithms that infer accurate breathing rate, as well as provide confidence scores for their outputs.
- Develop an algorithm that can infer the breathing rate from collected PPG signals.
- Package the algorithms and develop an ‘easy-to-use’ open-source framework that will allow researchers to input their own PPG data, chose a subset of our algorithms (or add their own), and obtain corresponding breathing rate output.

The purpose of this report is to document our investigation of algorithms for predicting breathing rate from PPG signals and to give recommendations for future directions of study. The report details the methods used to collect data along with algorithms that were found to predict breathing rate well and evaluation of their performance.

Though the high-level goal of our project is to aid overdose detection there are several limitations to the applicability of our investigation. Lack of access to PPG signals from real overdose patients required us to obtain data in artificial settings that simulate a decline in breathing from a very small group of healthy people (See Appendix C for details). As a result, it is difficult to make claims about the performance of our algorithms during real overdoses with a diverse group of people. In addition, because we collected our data without a proper housing, a real device might be able to pick up on additional information from the PPG signal that our algorithms do not have access to. Finally, although we initially set out to develop algorithms that would operate in real time, time constraints required us to develop algorithms that operate on windows of data, so additional work would need to be done in order to make these algorithms work in real time for the purposes of overdose detection. That being said, we also feel that the

framework would make it convenient for any researcher to continue to iterate and design new algorithms upon the arrival of more realistic data.

The rest of the report is organized as follows:

Discussion:
The discussion section begins with an overview of the opioid overdose and its symptoms along with how photoplethysmography is affected by breathing rate. It then goes on to give an overview of the experimental setup and data collection procedure. The section then illustrates the various features and algorithms developed as part of our project. Along with a high level overview of the framework that encapsulates these algorithms. Finally, we present the performance of various algorithms on the data we've collected.

Conclusions:
The conclusion section goes over the main conclusions that can be drawn from the analysis of the results presented in the discussion section.

Project deliverables:
The project deliverables section goes over the initial project deliverables, revisions to these deliverables, along with the final state of the revised deliverables. This section also goes over the projects financial summary, along with ongoing commitments from the team.

Recommendations:
The recommendations section gives an overview of recommendations for the future directions for the project, Along with justifications for these recommendations from the results of the discussion and the conclusions section.

Appendices:
The appendices contain additional information about the project. This includes:

- -Specifics of the data collection methods including wiring diagrams for the data collector.
- -Documentation on how to use the framework.
- -Code for collecting data from the Maxim photoplethysmograms.

Discussion

Theory

Opioid overdose is a toxicity due to excessive opioid intake. The main symptoms include insufficient breathing, shrunken pupils and unconsciousness. The primary danger in opioid overdoses is the depressant effect of opioid agonists on respiration. These drugs can directly depress regions of the brain responsible for respiratory control and lead to dangerous levels of respiratory depression even at opioid dosages that do not disturb levels of consciousness. The amount and characteristics of the respiratory depression can vary greatly by dose and by person. Death from opioid overdoses is almost always a result of respiratory arrest [1].

During an opioid overdose, respiratory depression affects both the rate of breathing as well as the amount of air intake. The respiratory rate is defined to be the number of breaths one takes per minute. A normal respiratory rate is 12 to 20 breaths per minute(bpm) at rest. However during an opioid overdose a respiratory rate of 5 to 6 bpm is common. Breathing rate is one of the clearest quantitative indicators of opioid overdoses and a breathing rate of below 8bpm is indicative of an overdose [2]. Tidal volume is the lung volume representing the normal volume of air displaced between normal inhalation and exhalation when extra effort is not applied and is also reduced. In addition, the blood oxygen saturation will also be reduced by the respiratory depression and a level of 92% or be low is indicative of an overdose [2].

A pulse oximeter is a sensor that uses spectroscopy to measure peripheral oxygen saturation. In our project a reflectance based pulse oximeter (referred to as the ‘Max’ sensor) is used which reflects multiple wavelengths of light onto the patient's finger, measuring the resulting difference in light absorption which is due to the light absorption properties of oxygenated blood. Monitoring this signal over time results in a photoplethysmogram (PPG).

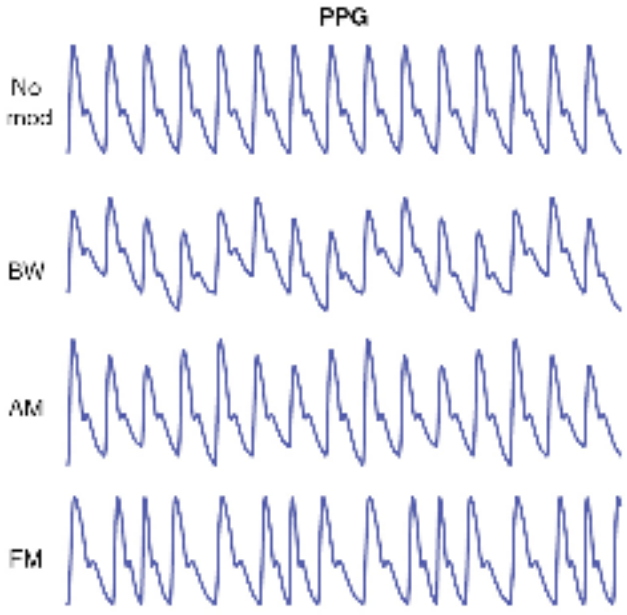


Figure 1:Illustration of PPG Signal Modulation[6]

Although each cardiac cycle is visible by its peaks and is the most prominent frequency in the PPG signal, the signal is also modulated by breathing. As a result, it’s possible to use a PPG to monitor a patient’s respiratory rate. A review of the literature reveals that the PPG signal can be modulated in three prominent ways:

1. There is an underlying low frequency component in the PPG that lines up with a patient's breathing. This is often referred to in the literature as BW (Base Wave) modulation.
2. There is variability in the amplitude of the PPG signal that also correlates with the breathing rate. This is usually referred to as AM (Amplitude modulation).
3. The frequency of the heartbeat in the PPG is also modulated by the breathing cycle. This is often referred to as FM (Frequency modulation).

See Figure 1 for an illustration.

Although each form of modulation has been shown to exist, the presence of any given form of modulation varies from patient to patient, and as a result many approaches in the litterature use a combination of methods to detect the breathing rate.

Preliminary Analysis

The central problem of our first objective is an inference problem; ie, we want to provide an accurate mapping from PPG signal to the corresponding breathing rate. For this, we first explored various features of the signal to determine which features would be most relevant.

Because we did not initially have access to the hardware to collect our own data, we decided to explore the commonly used ‘Capno’ database, which is a public dataset of PPG signals from patients under anaesthesia [4].

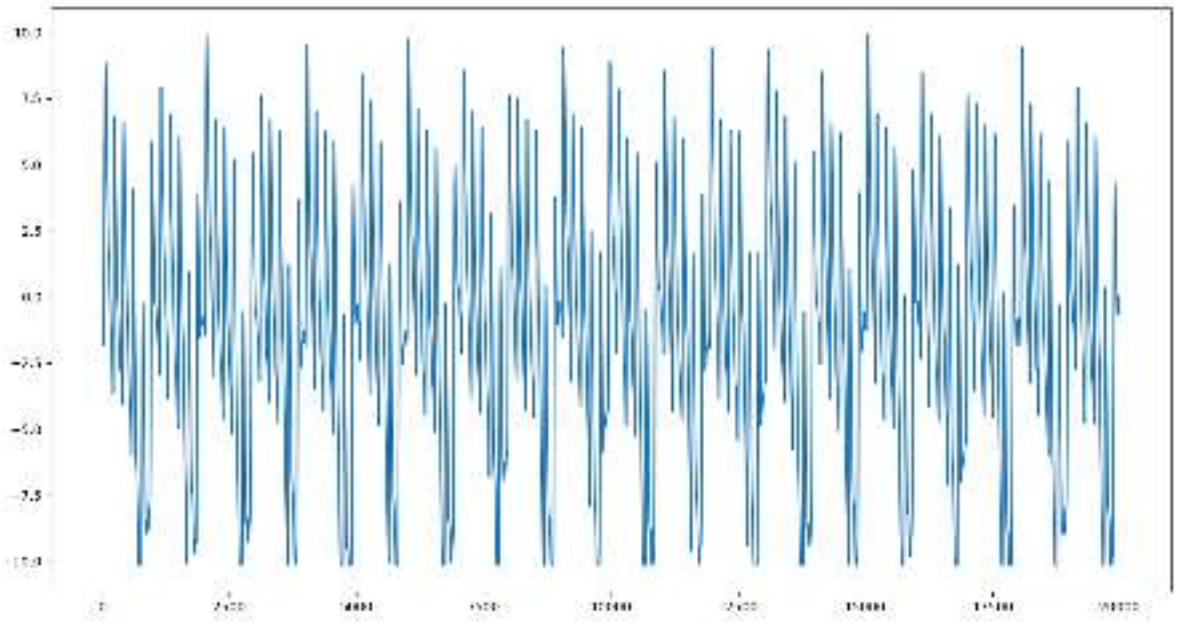


Figure 2a: Sample 20 second long PPG signal from the Capno Dataset.

Because we wanted to extract the breathing frequency, we looked at the signal in the frequency domain first.

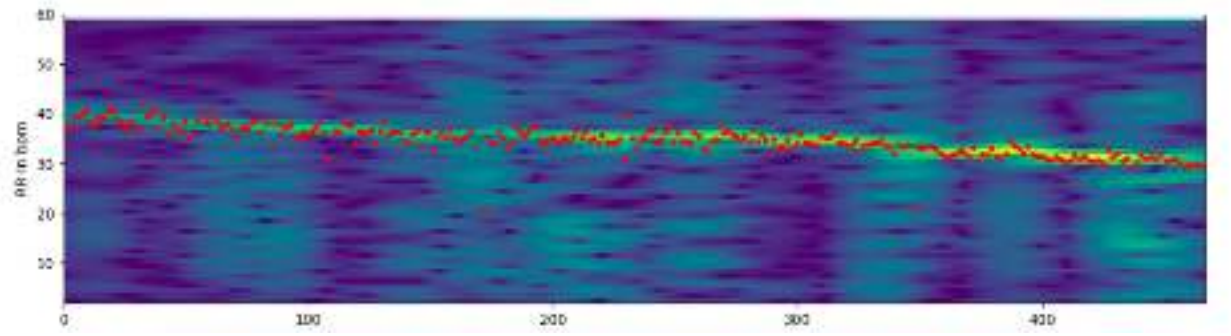


Figure 2b: Short-time fourier transform (background) and labelled breathing rate (red dots) on a capno sample. The labelled breathing rates clearly coincide with bright regions in the fourier spectrum.

From our analysis of the capno signal, we made two observations:

- It shows little variation in breathing rate and is simple to capture in the fourier domain
- The signal the breathing in highly irregular and the task of predicting breathing rate is nearly impossible

Furthermore, a lack of variation in breathing rate is seen across most patients in the dataset, with a few exceptions

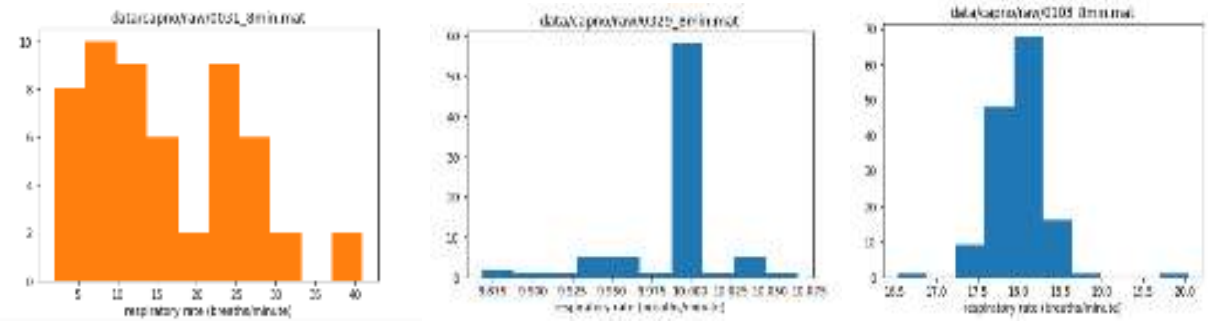


Figure 2c: Histogram of breathing rates across samples of different patients

Data Collection

Although the Capno dataset was a good starting point to gain intuition about PPG signal processing, it was crucial to perform well in a ‘real-world’ scenario with the max sensor that the vitalsigns team will be using. Because we did not have access to opioid overdose data, we collected simultaneous PPG data and thermistor data from healthy individuals. To do this, we used hardware provided by other ENPH teams working with the Vitalsigns group.

We recorded data under a variety of circumstances including but not limited to (See Appendix C for exact experimental details):

1) Normal breathing
2) Not breathing
3) Slow breathing
4) Sudden transition from regular breathing to slow breathing
5) Sudden transition from regular breathing to stop breathing

The experimental setup was as follows:
With the aid of a rubber band, an individual's finger is tied to the max sensor and a thermistor is also added around the individual's nose. Readings from both the max sensor circuit as well as the thermistor are input to an Arduino Uno, the output of the uno is then written to a Comma Separated Values file on a computer.

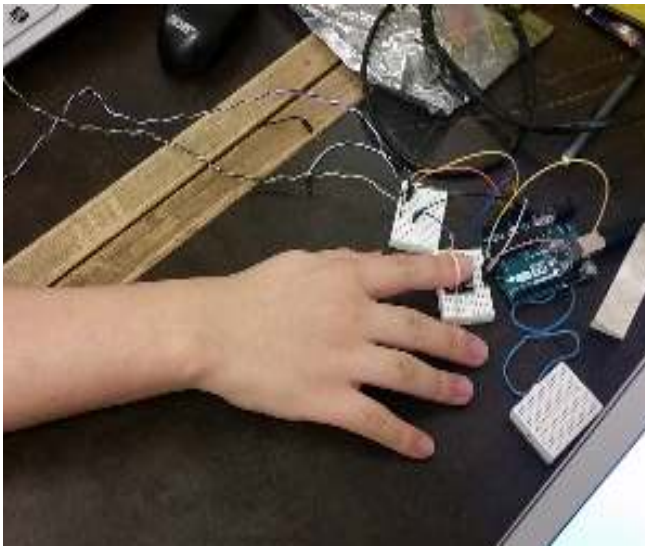


Figure 3a (left): Collecting data from subject; the thermistor (green) is attached below the nose while finger is placed on the Max Sensor

Figure 3b (right): A subject placing his finger on the max sensor (held by a rubber band), which inputs readings into the Arduino Uno, which are then recorded on a computer.

See Appendix A for the pinout diagrams of the max sensor, thermistor, as well as links to the code for reading data from the Arduino.

Preliminary Analysis of the Collected data

Here is one sample of ‘normal breathing’ data

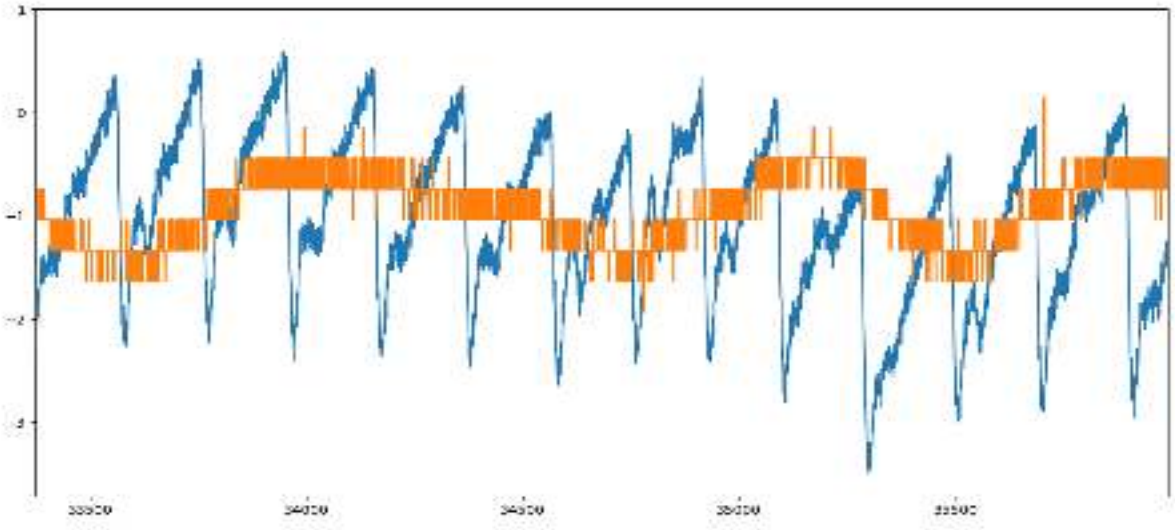


Figure 4a: exp_009 sample of the normalized (mean 0 standard deviation 1) but unfiltered ppg signal vs sample number. PPG signal is in blue, thermistor is in orange

We see that the PPG signal from collected data, when compared to the capno PPG signal (Figure 2a and Figure 2b), is much noisier. By further analyzing the fourier spectrum of the PPG signal that we collect, we see that a clear breathing rate is much harder to track in the fourier spectrum (see figure below). However, for the thermistor, the breathing frequency is clearly visible and this signal shows less noise than the corresponding PPG signal.

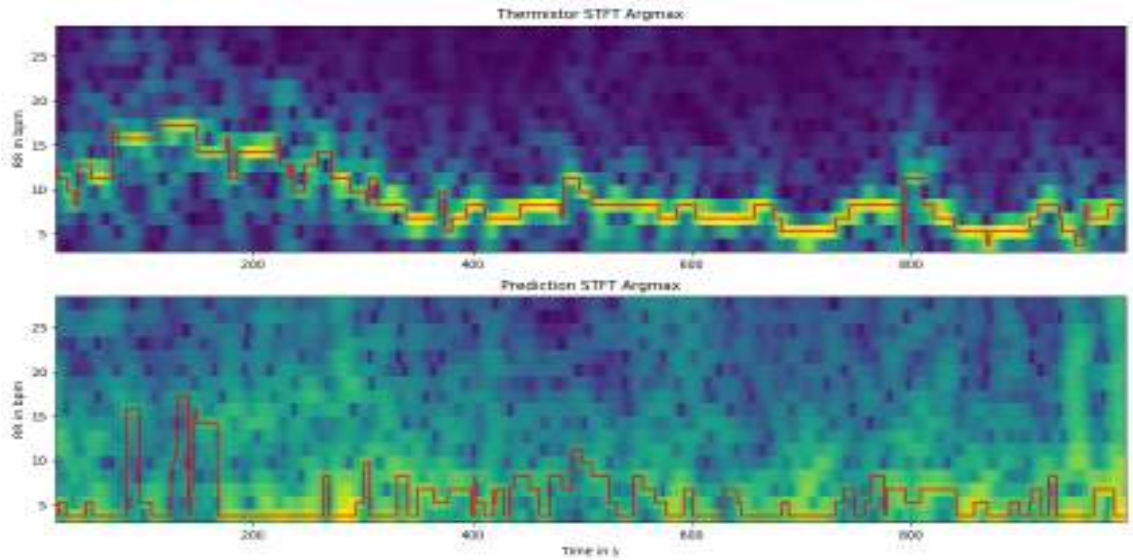


Figure 4b: Short-time Fourier Transform on a raw PPG signal and corresponding filtered thermistor signal. We see that the strongest frequencies (up-to 25 bpm) in the PPG signal don't correspond to the strongest frequencies. Thus a more in-depth analysis is needed.

Whereas the capno signal showed little variation in breathing rate, PPG signal in our data includes additional noise from variation in finger pressure on the max sensor

Preprocessing

The signals from both the max sensor and the thermistor are both very noisy as a result we eliminate unwanted noise and variations in the means of both signals using the following methods:

Local Normalization: To eliminate variations in both the mean and standard deviation of a signal. We compute the average and standard deviation around a point, subtract the computed mean and then divide by the computed standard deviation. Example results are shown below.

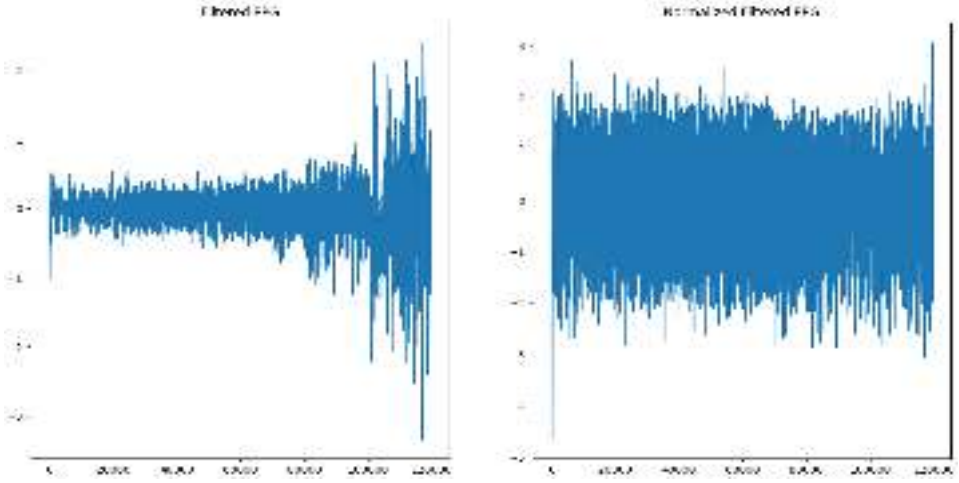


Figure 4c: PPG signal on the left, PPG signal after local normalization on the right

Butterworth Filtering:
To efficiently smooth the signal, we apply a butterworth filter bandpass filter to a signal. Although this method is very fast, it has the adverse effect of inducing a phase shift in the processed signal.

Spline Interpolation:
To smooth the signal without inducing a phase shift, we use regularized spline interpolation in order to eliminate high frequency components in a signal. Due to computational constraints, we downsample the signal.

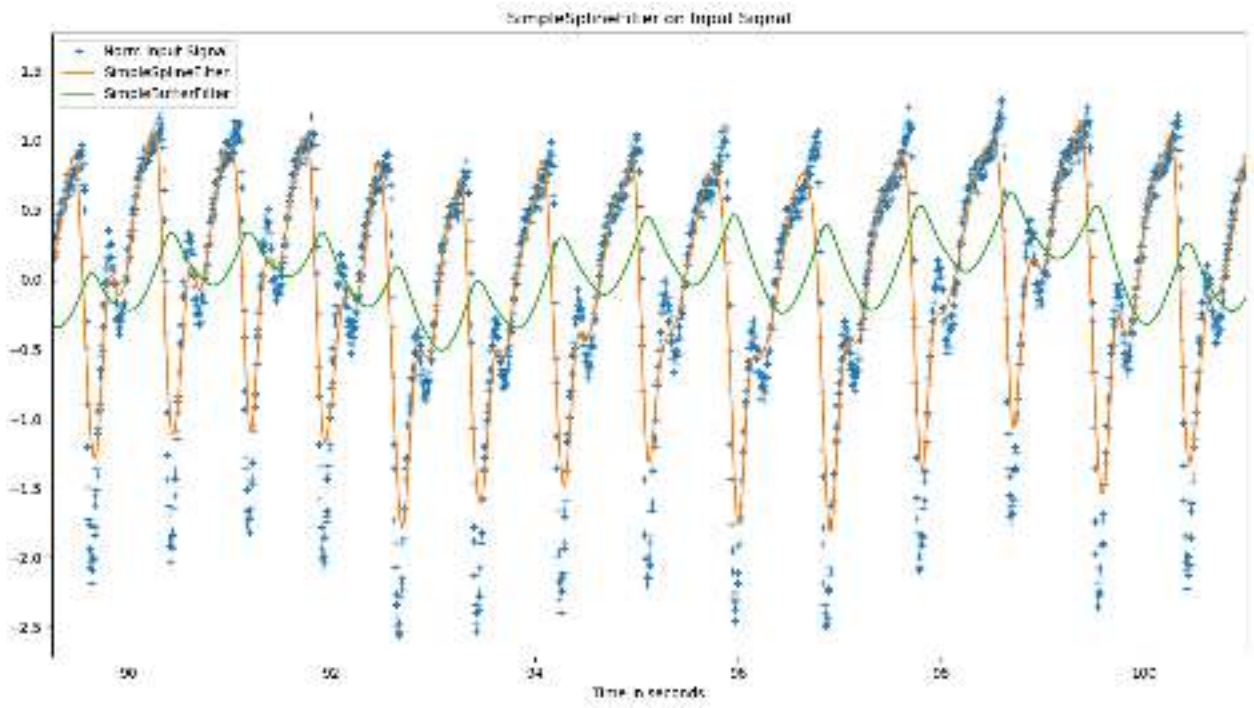


Figure 4d: Comparison of the effect of regularized spline interpolation and butterworth filtering. The blue points represent the raw signal, the green indicates the output of the butterworth filter, and the orange indicate result of spline interpolation. Although the butterworth filter is faster to compute, it alters the amplitudes of the incoming signals and induces a phase shift in the signal.

Preprocessing for the PPG signal:

As seen in Figure 4a, the PPG signal from the max sensor is very noisy. Whereas the capno dataset showed little variation in its mean, PPG signals in our data includes additional noise from variation in finger pressure on the max sensor. To account for this variation, we developed a method that applies local normalization over the PPG signal. This step removes the variation resulting from changing finger pressure.
Next we apply a 3rd order Butterworth bandpass filter from 3 bpm to 90 bpm. The goal of this step is to eliminate irrelevant frequencies.

Preprocessing for the Thermistor signal:

Because the thermistor more clean (and directly) captures breathing rate, we are mainly interested in removing noise and discretization artifacts from the signal. We also want to remove variations in the mean of and standard deviation the signal (unlike the PPG signal, where we also want to filter out certain frequencies). We thus pre-process the thermistor as follows:

1) Subsampling of the 200Hz signal at 10Hz
2) Spline interpolation on this subsampled signal to remove noise
3) ‘Local-norming’ the signal to remove signal mean variation

Feature extraction

Here we outline the features that were extracted from the underlying PPG signal. Though we want to ultimately capture breathing rate from the PPG signal, we want to first find features that correlate well with the thermistor(breath) signal. This step allows us to more effectively do further processing later on in order to predict the breathing rate. After preprocessing the PPG signal to remove irrelevant frequencies and mean variations, we compute the following features for the PPG signal:

a) Envelope Amplitude (EA):

As stated in the theory section it is known that the frequency of the envelope of the PPG signal correlates well with the breathing rate. We first capture this envelope with a peak-detector, identifying both the peaks and troughs of the signal, as shown below.

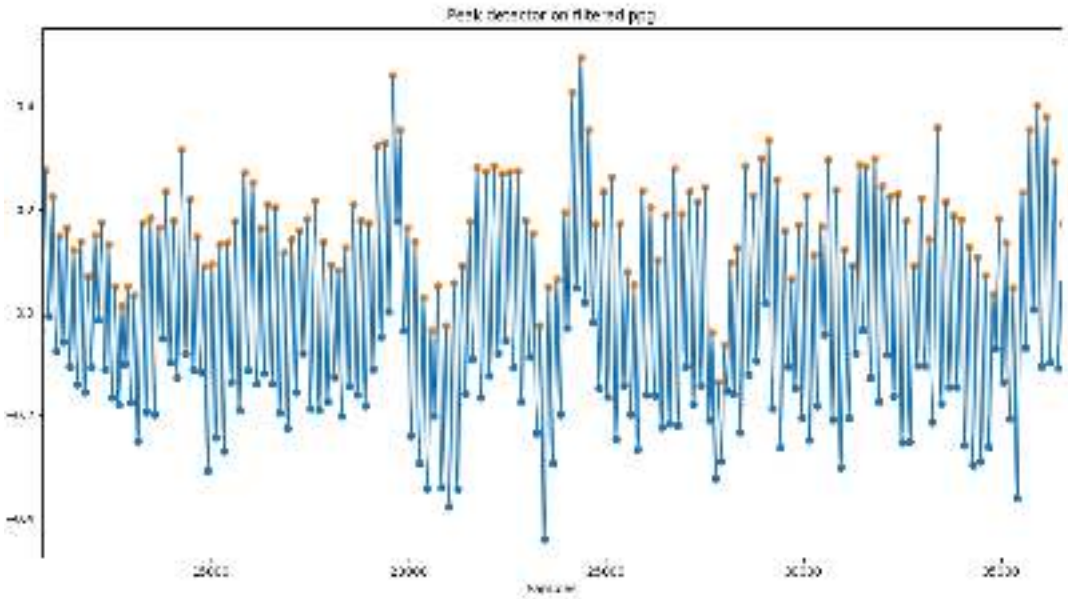


Figure 5a: Peak detection on sample PPG signal. Orange points are detected peaks and blue peaks are detected troughs.

We can then spline-interpolate between these peaks to obtain the signal envelope of the PPG.

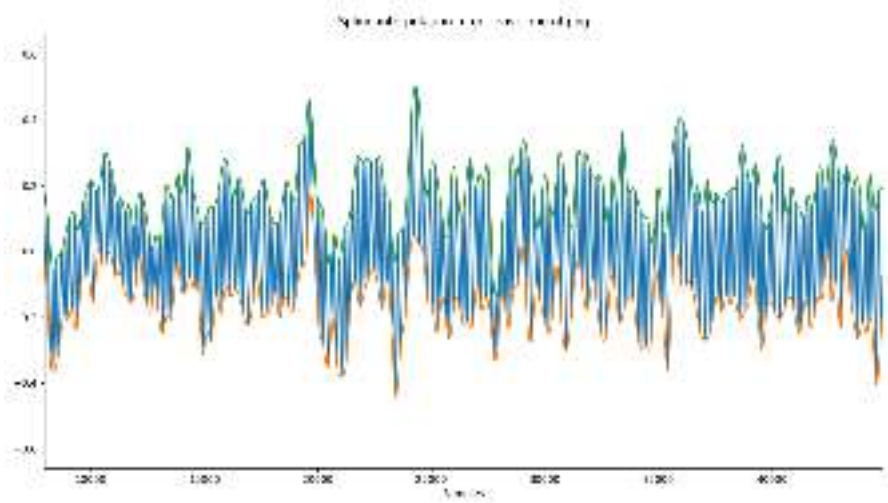


Figure 5b: Interpolation performed through the peak and trough points respectively on sample PPG signal.

Instead of looking at the upper or lower envelope alone, we subtract them to obtain a more robust proxy for the amplitude of the envelope (since otherwise, the signal is misleading in cases where the top or bottom envelope changes drastically).

As seen below, this feature shows a strong correlation with the corresponding thermistor signal, except in some regions where the envelope amplitude shows two peaks instead of one.

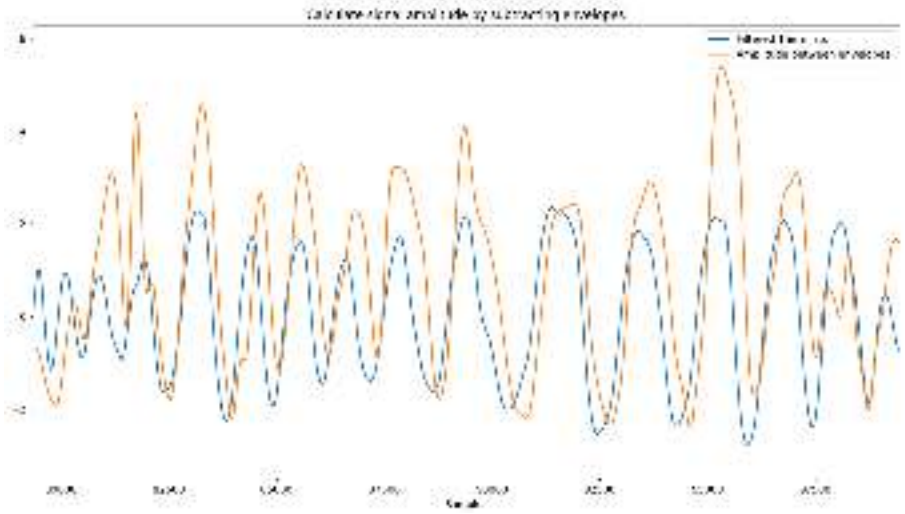


Figure 5c: Envelope Amplitude on sample PPG signal. This correlates well with the thermistor data.

b) PP (Peak Period), TP (Trough Period)

We also know that the frequency of the heartbeat is modulated by the breathing rate. To extract the frequency of the heartbeat, we again find peaks and troughs in the PPG signal, as shown for

the feature above. For Peak Period, we compute the period between consecutive peaks, and then perform spline interpolation, as shown below. There is clear correlation between this feature and the thermistor signal.

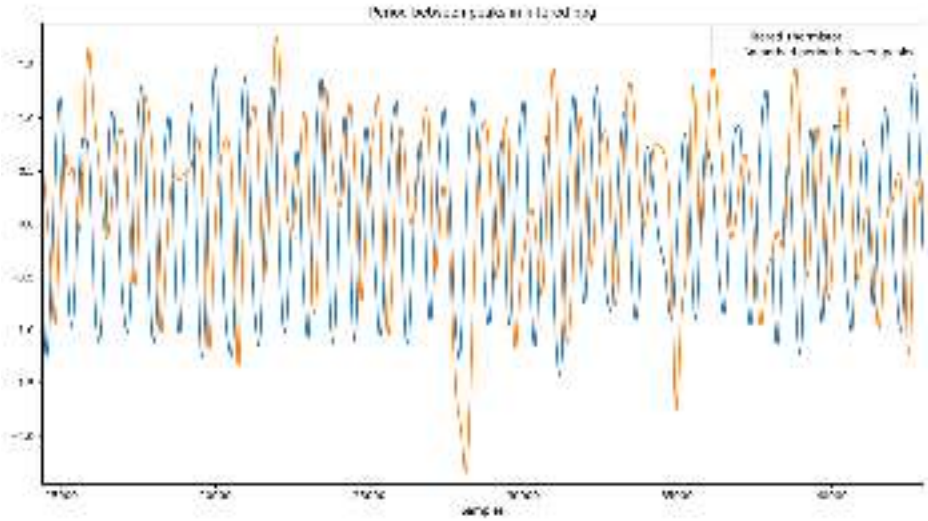


Figure 5d: Period between peaks Feature. This feature also shows correlation with the thermistor signal.

Similarly, for Trough Period, we compute the period between consecutive troughs of the PPG signal, and then then perform spline interpolation.

c) Convolutional Neural Network

In order to further improve breathing rate prediction accuracy and to ensemble the features that we've extracted, we chose to use a convolutional neural network that takes extracted features and outputs a prediction of the corresponding thermistor value at the same point in time. The network operates by alternately convolving kernels over the features and then applying a non-linearity to said features. The output of the CNN can therefore be thought of as another feature (however, this feature is learned) that predicts the breathing rate. A maximum likelihood estimate is used as the objective function to align the CNN's output with the thermistor. This objective function is then minimized via backpropagation. See [5] for a more detailed explanation of the learning procedure for convolutional neural networks.

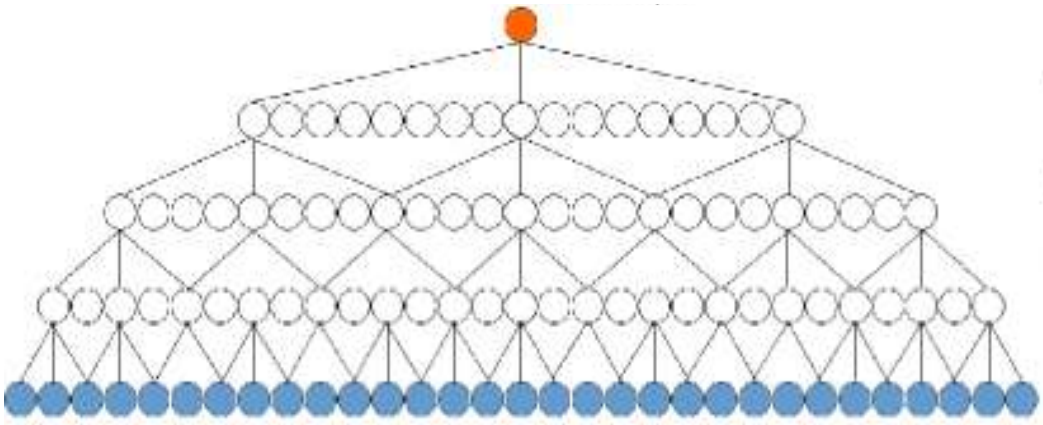


Figure 6: A standard convolutional neural network architecture illustration.The blue nodes represent the input signal, and the edges represent the weight parameters. The white nodes are obtained after convolving the layer underneath with a convolution kernel.

An interesting observation regarding the CNN is that in order to get it to work we needed to heavily downsample the data by a factor of ~12 in our experiments. We found that using all of the data (without downsampling) led to over-fitting, where the CNN would also ‘learn’ the noise in the training set, so the downsampling helped the CNN to generalize to previously unseen data.

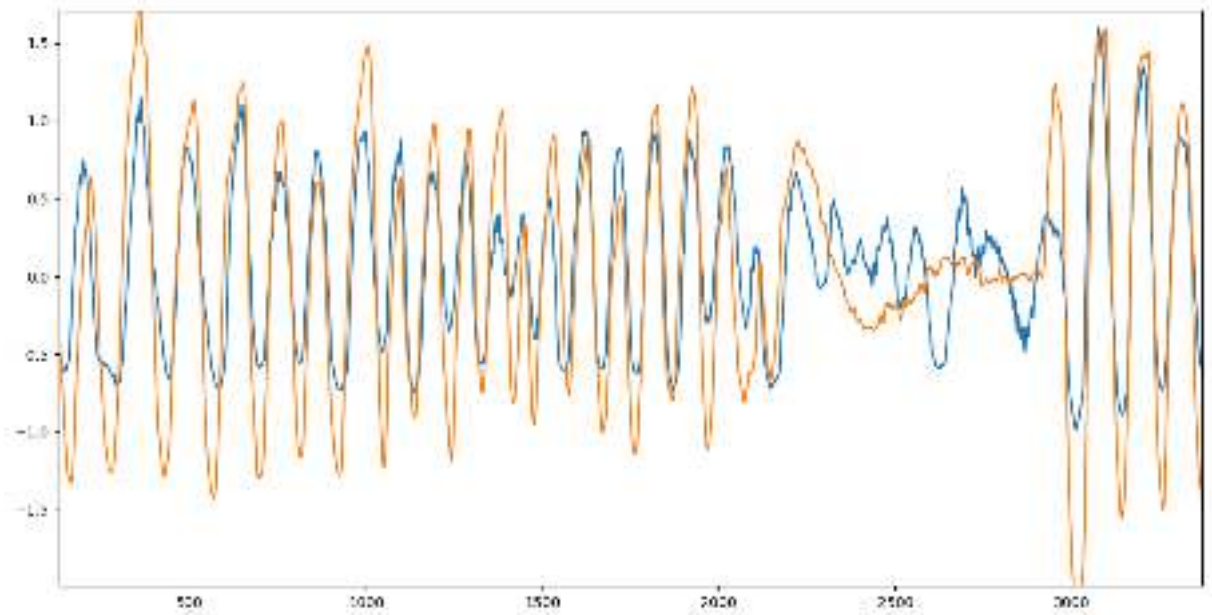


Figure 7: The predicted thermistor value from the CNN on experiment 10 after being trained on the data from experiment 11. Data has been downsampled by a factor of 12 before being fed into the network. Both signals are normalized. Predictions are in blue, true breathing rate is in orange. The subject was holding his breath roughly between samples 2250 and 2900.

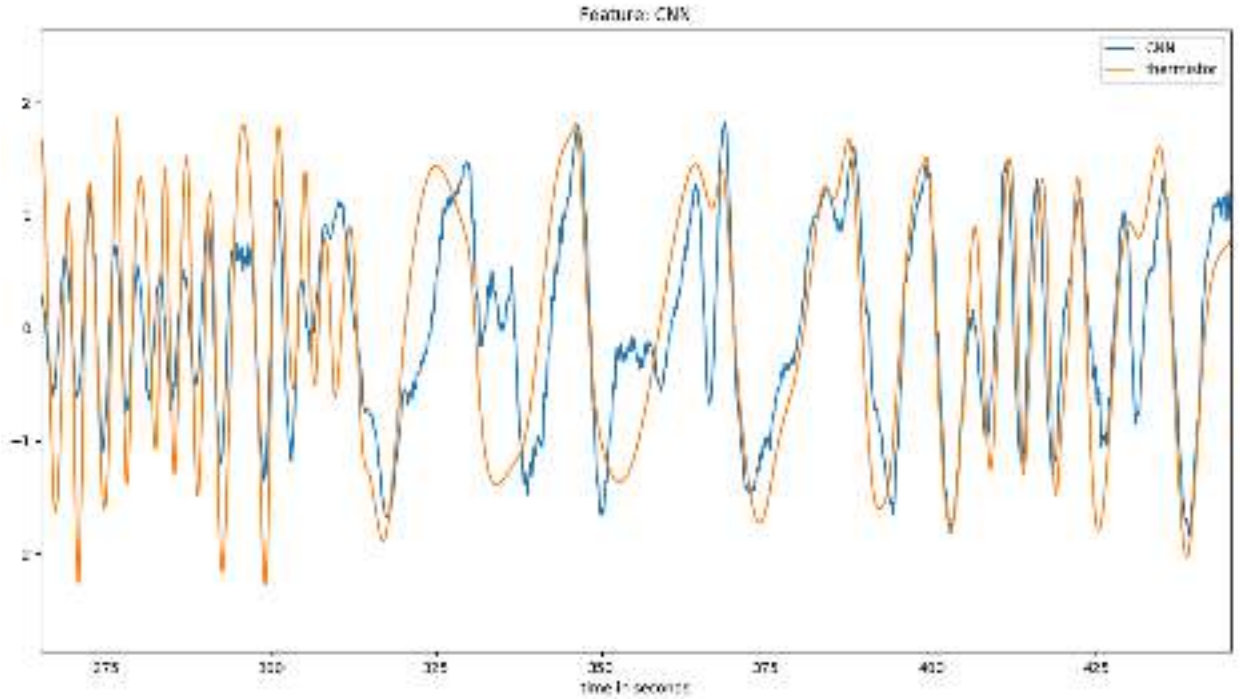


Figure 8: Failures of the CNN. The CNN will occasionally fail to capture breaths, or it will believe that multiple breaths occur within one period. The first effect is clearly visible just after 300 seconds in the figure below. The second observation is especially clear immediately after 425 seconds. Another interesting note is at the same time as the CNN predicts a large dip, there's often a small dip in the thermistor value, this is clearly visible at around 365 and 435 seconds.

Breath Prediction

After extracting signal features, either a feature or a thermistor like signal from the CNN. We've developed two algorithms to predict breathing rate from the PPG signal

Instat BPM Peak Detection: The simplest method to detect breathing rate from a PPG signal is to simply filter the input signal and detect peaks in it. From there, we can compute the period between the peaks and take the reciprocal in order to estimate the breathing rate. This method has an advantage of being very accurate when the underlying feature faithfully tracks the thermistor signal up to a phase constant.

STFT Centroid Breath Frequency Estimation: When the underlying feature is not as clean, it's better to apply a local fourier transform algorithm to the signal. This yields a 2 dimensional array of values that describe the signal, one dimension corresponding to time and the other to frequency coefficients. We take the element wise power p of the STFT and the take the resulting centroid along the frequency axis at each point in time. This yields an estimate of the breathing rate. When p = 1 this corresponds to a finding the centroid in the frequency spectrum. When p tends to infinity this corresponds to taking the largest mode of the frequency spectrum. This

method has an inherent disadvantage in the form of a time frequency resolution trade off. However, it is more robust to noisy features.

PPG Breath Prediction Algorithm

The overall algorithm pipeline is shown in the diagram below:

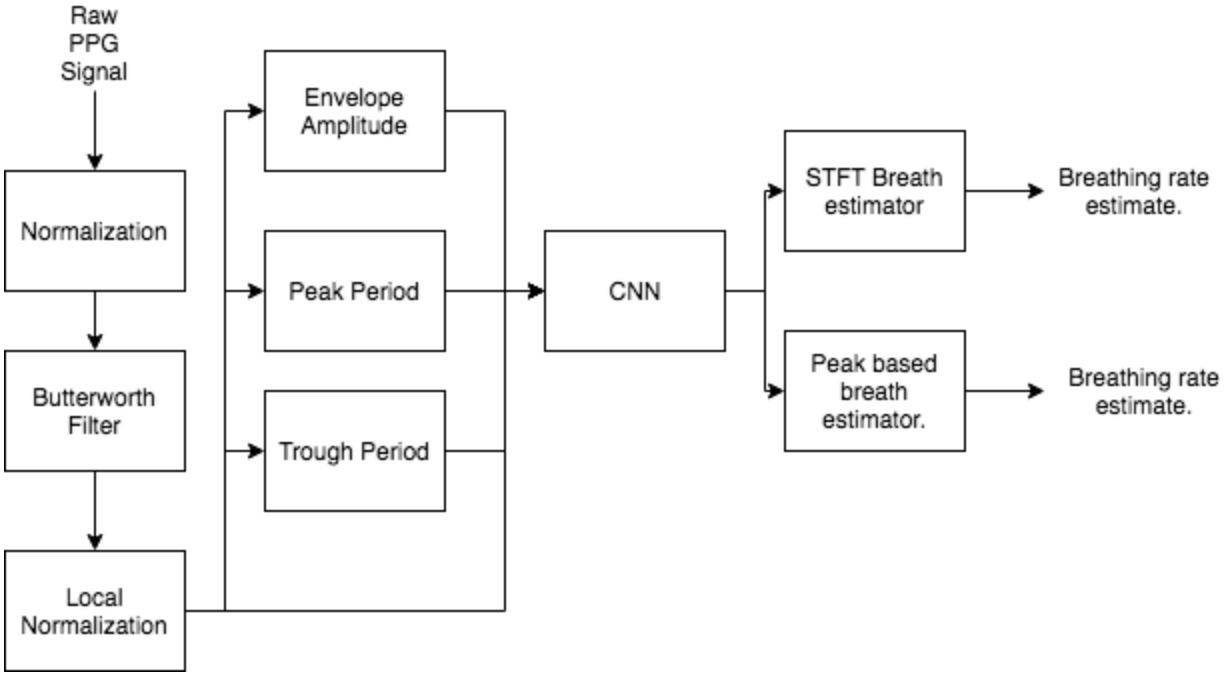


Figure 9: Flow diagram of the algorithm. The first step is pre-processing, followed by feature extraction, followed by the features being input to the CNN which outputs the thermistor estimate. We then extract the breathing rate using two different methods.

Framework

In order to make it easier for the project sponsor to train the algorithms we've developed on new data. We've packaged all of our algorithm implementations into a framework that makes it easier to train new algorithms. Each feature supports a common interface which makes it possible for the features to be chained together into a model.

The framework takes in a JSON description of the features of the model, along with testing and validation data sets. It then builds the model according to a specification and trains each component sequentially.

Each feature has 4 properties associated with it. The first is a string designating the name of the feature, so it can be easily referenced later with a dictionary. The second is the list of features

that this feature takes in as input. The third is it's type, which indicates which algorithm it implements. The fourth is a set of key-value pairs indicating the parameters of the algorithm.

Results

We first compare the performance of the CNN with that of the EA feature alone, to ensure that addition of the CNN is not worsening performance. We find that although both signals generally capture the breathing frequency, the CNN performs better in some edge cases. Below is a representative comparison of the CNN and EA on a sample PPG signal. The same observation is made across other samples.

We find similar advantage in using the CNN when compared with other features, ie. the CNN is able to perform slightly better than each of the features that are input to it. This confirmed our hypothesis that the CNN would be able to 'fuse' its input to perform better than each of the input features.

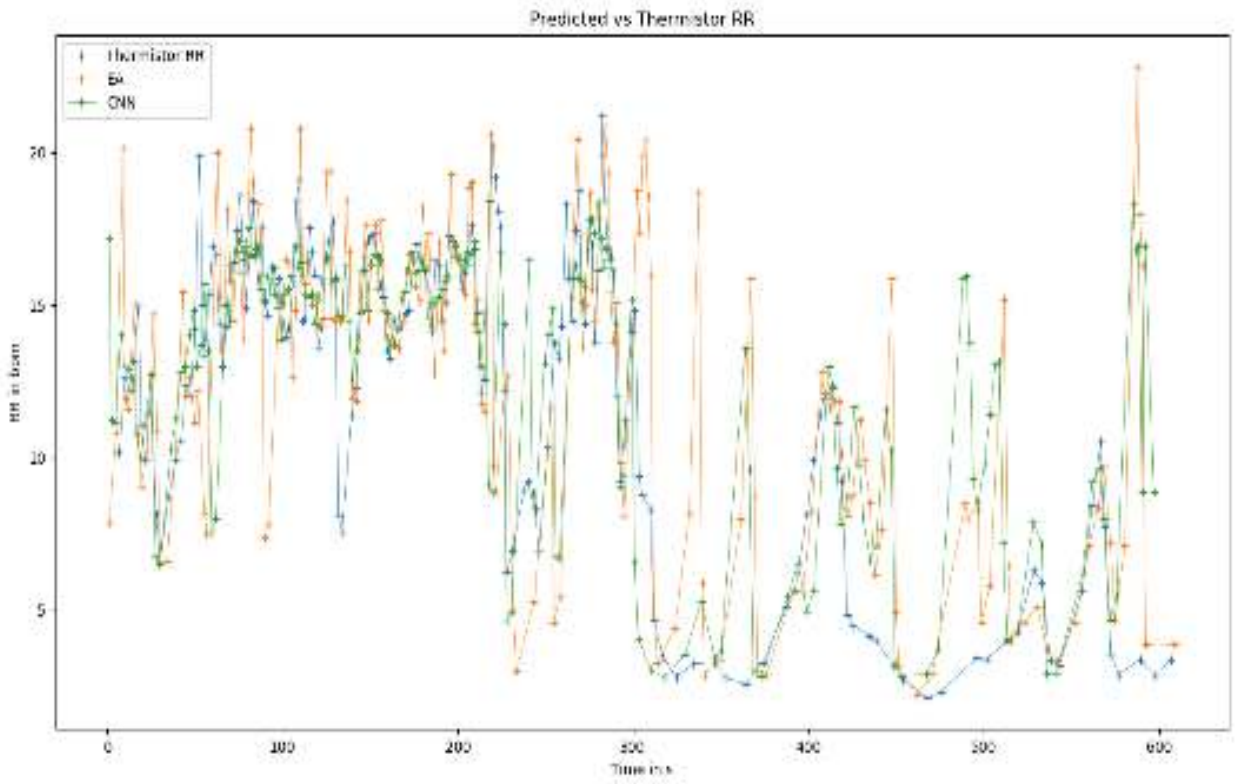


Figure 10: Comparison of EA vs CNN. The points in the figure are instantaneous breathing rates calculated by Instant BPM for EA (orange), CNN (green) and Thermistor (blue). We see that the CNN is better able to track the thermistor's signal overall, avoiding extra high bpm breaths compared to the EA feature.

As is common with machine learning workflows, we split our data into a ‘train’ set and a ‘validation’ set. We train the algorithm on the ‘train’ set, and validate its accuracy on the ‘validation’ set.

Given the limited amount of data we are able to collect, we cannot simply use a fixed train and validation set when evaluating our algorithm as this would shrink our training data too much or leave use with a small validation set. This is a common problem in machine learning in practice and the common solution is to use K-Fold cross validation, whereby the model is re-trained and evaluated for different train validation splits. In our case the validation set consists of single signals that are representative of the whole dataset. In the table below, we list the algorithm’s performance when the validation set is as listed and training set is everything other than the validation set (except in run 4 where we remove another sample from the training data). Note: the details for data from each experiment are listed in Appendix C.

Run	**Validation Signal**	**CNN Centroid RMSE**	**EA Centroid RMSE**	**CNN Instant RMSE**	**EA Instant RMSE**
1	Exp 31: Curtis transition to slow breathing.	***1.1449664872579712***	2.139095224951521	***2.488413886670904***	4.293727311057002
2	Exp 26: Rahat transition to slow breathing.	***2.0686025177907172***	2.2939524187275286	***3.702359835350327***	3.744102051767237
3	Exp 9: Curtis normal breathing.	***1.3313603536831529***	2.0373375870299864	***3.136755142690628***	4.036406044734868
4	Exp 26: New subject generalization. (exclude exp 27 data in train)	***2.014204383958743***	2.2939524187275286	***3.293793111204684***	3.744102051767237

Table 1: RMSE results. Root mean squared error (RMSE) computed between the extracted RR estimates using the instant bpm and STFT centroid methods on the CNN and the EA models. Validation signals were selected in each run to represent the whole dataset.

We summarize our algorithm’s performance on each of these runs.

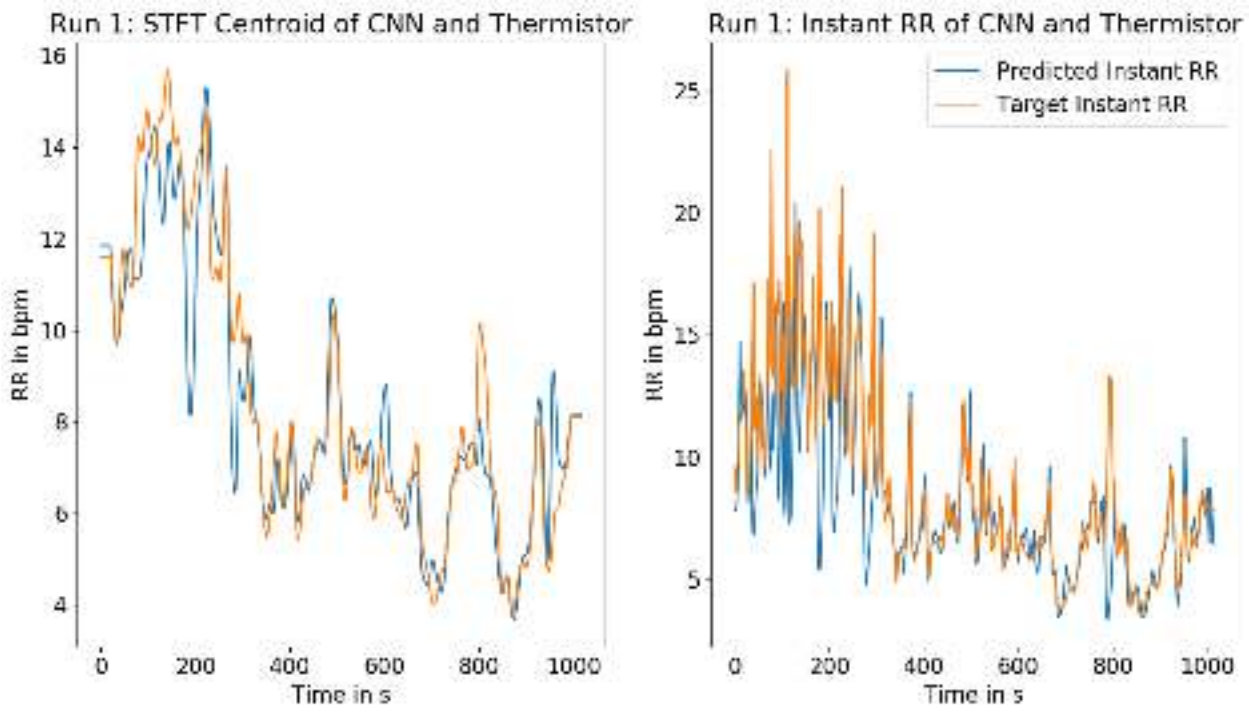


Figure 11a: Plot of the extracted respiratory rate estimation using STFT centroid and instant RR of the CNN on Run 1. Run 1 uses experiment 31 as the validation signal. This dataset consists of a short period of normal breathing followed by a period of slow breathing. The data was collected from subject Curtis.

As can be seen above, the algorithm works very well in the slow breathing portion after 350 seconds. The large variability in respiratory rate in the normal breathing section in the first portion is common for regular breathing and makes it exceedingly difficult to estimate the instant RR. Instead the CNN performs very well in terms of the STFT centroid breath extraction.

Run 2 uses experiment 26

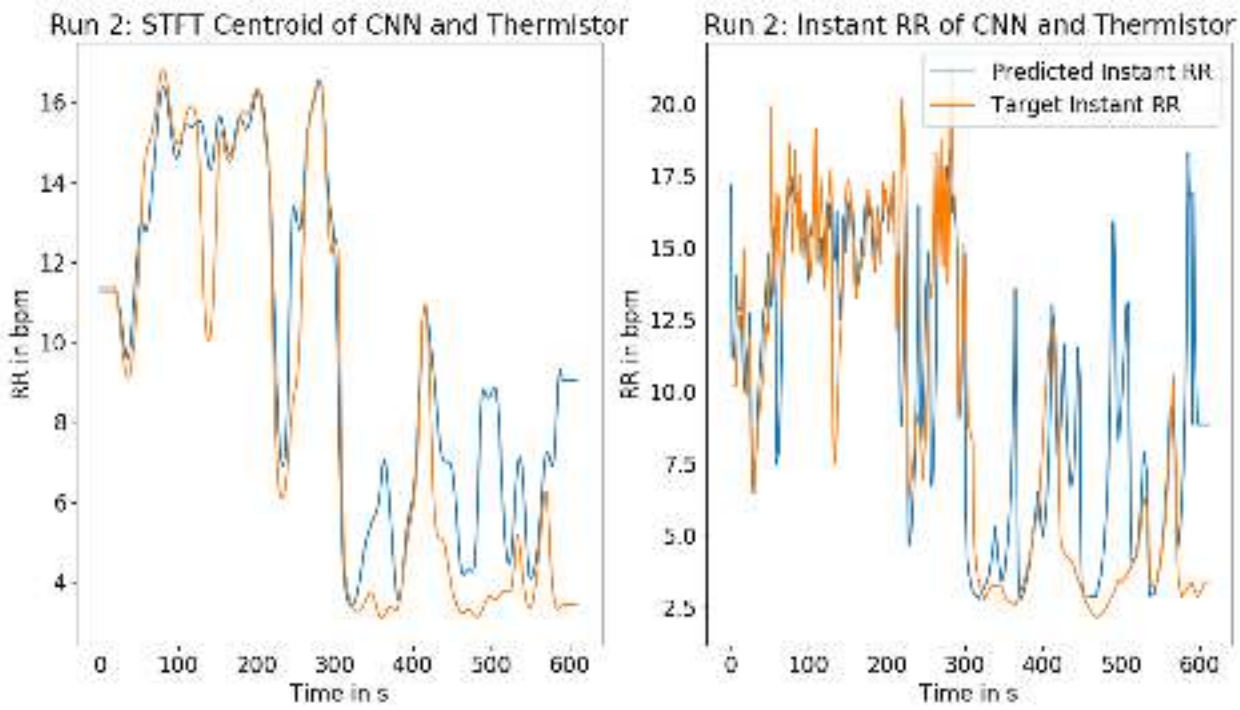


Figure 11b: Plot of the extracted respiratory rate estimation using STFT centroid and instant RR of the CNN on Run 2. Run 2 uses experiment 26 as the validation signal. This dataset consists of a short period of normal breathing followed by a period of slow breathing. The data was collected from subject Rahat.

In the figure above, we see sudden speeding up in breathing in the slow breathing section. It is likely that these spikes in the true breathing rate also lead to the erratic spiked breathing rate predictions by the CNN caused by double breaths. The sudden drop and transition in breathing rate is captured well by the CNN.

Run 3:

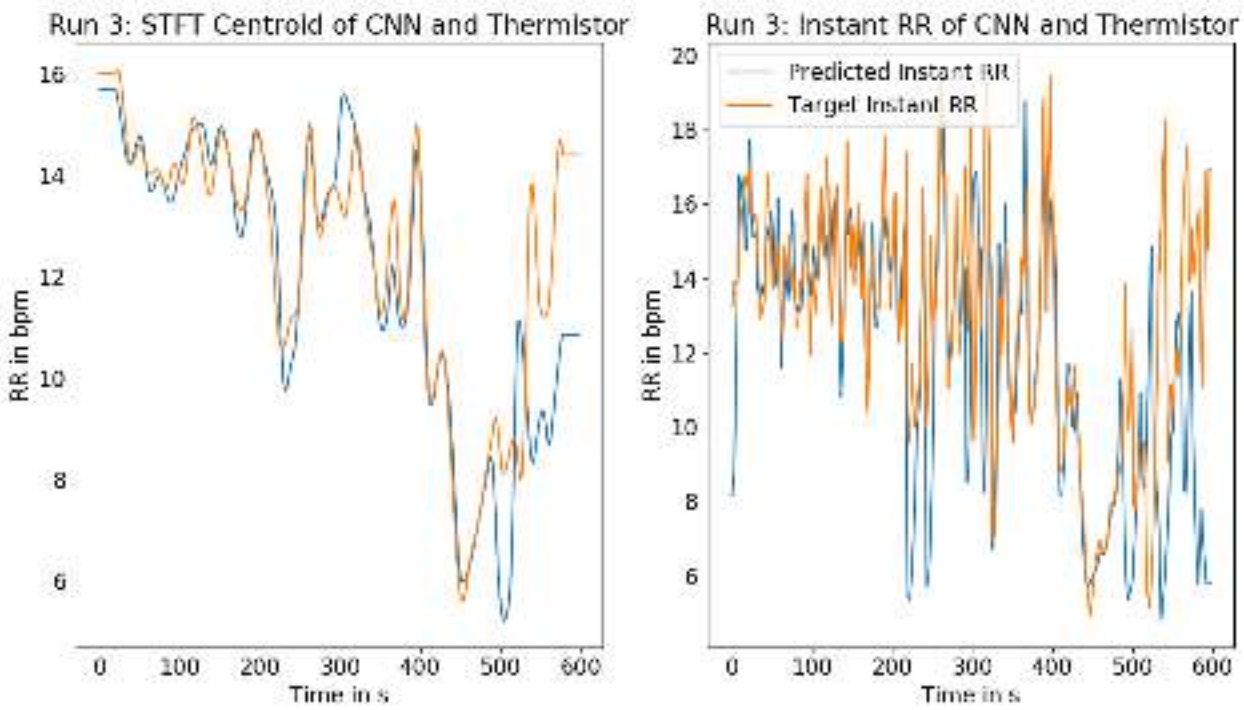


Figure 11c: Plot of the extracted respiratory rate estimation using STFT centroid and instant RR of the CNN on Run 3. Run 3 uses experiment 9 as the validation signal. This dataset consists of a short period of normal breathing. The data was collected from subject Curtis.

In run 3, we illustrate the performance of our algorithm on a normal respiratory rate signal. The performance of the CNN on this signal is quite high compared to signals where the subject deliberately slows down their breathing rate, such as in Run 2. It may be the case that deliberate slow breathing has a different effect on the PPG compared to involuntary slow breathing. We are unable to confirm this as it is difficult to collect longer periods of involuntary slow breathing.

Run 4:

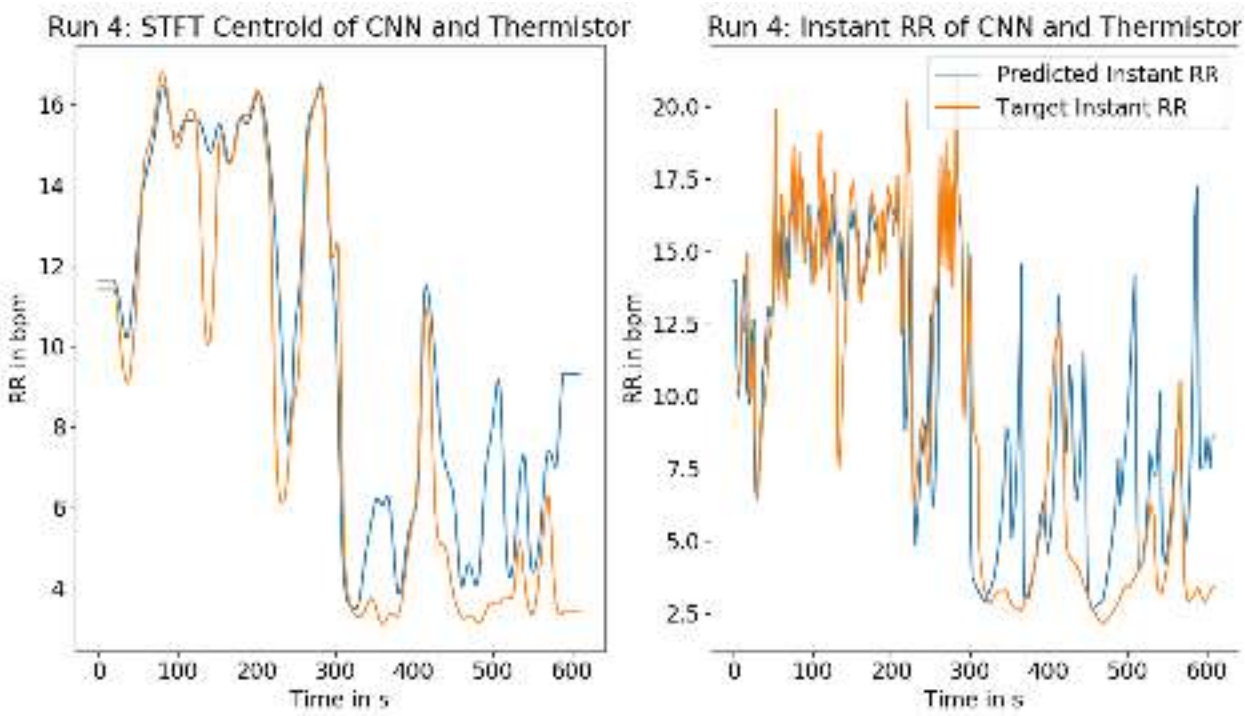


Figure 11d: Plot of the extracted respiratory rate estimation using STFT centroid and instant RR of the CNN on Run 4. Run 4 uses experiment 26 as the validation signal, but in addition does not include experiment 27 in the training set. This dataset consists of a short period of normal breathing followed by a period of slow breathing. The data was collected from subject Rahat.

In run 4, we were interested in testing the CNN's ability to generalize to new subjects. To do this we use the same validation signal as in run 2 but we remove all signals associated with this subject from the training set. This way the algorithm is validated on a signal collected from a subject that it has not seen before. Qualitatively, there is almost no difference between the predictions of the CNN on run 2 and 4. One might expect an increase in the RMSE as the CNN would not be trained with data associated with the subject. Alternatively if the loss did not increase much, it would suggest that the CNN has the ability to generalize to new subjects. Interestingly, the RMSE *drops* a little bit for run 4 compared to 2. Although the drop in RMSE is not expected, it is not a significant amount and may be due to variance. However this at least suggests that the CNN is able to generalize to new subjects, given that it attains a high level of performance on never before seen subjects.

Across all the runs, the CNN only captures the instantaneous breathing rate well on average. It is unable to capture the sudden spikes that are seen in the instantaneous breathing rate. This is not problematic since our objective is not to capture every single breath correctly. We are more concerned about capturing the average breathing rate over a window of a few seconds.
In addition, and importantly, the breathing rate is captured well even at lower frequencies and in particular, the breathing rate can be tracked *while* it is decreasing; so the algorithm could be useful in a real overdose setting.

The STFT Centroid plot more aptly reflects the ability of the CNN to smoothly capture the mean respiratory rate over a window since the STFT frequencies are computed over a 40-second window.

Generally speaking, the algorithm does well in tracking the instantaneous breathing rate within +/- 2.5 - 3.5 bpm, as reflected in the root-mean-squared-error. This error is sufficient to characterize if the breathing rate is increasing or decreasing and is comparable to performance reported in the literature albeit on other datasets.
Additionally we report an RMSE of +/- 1.1 - 2.1 bpm on our STFT centroid breath extraction method which is more representative of the overall breathing rate and is a better indicator for performance in the target application.

Conclusions

The algorithm that was developed shows promising results in inferring breathing rate from PPG signal on healthy patients, when breathing is normal as well as when breathing is slowed-down. Although the breathing rate error is non-negligible (of around 3bpm in our data), the algorithm is able to capture the *change* in breathing rate quite well. This ability to track the change is crucial to determining whether an overdose is occurring.

It's important to once again highlight the limitations of our analysis and methods. The data that we collected includes instances of *simulated* slow-breathing, which is not necessarily similar to the target problem of an opioid overdose. For example, the heart rate goes up when breathing slowly artificially, but there is no reason to believe this is the case when breath slows down during an overdose. In addition, all of our data comes from 5 college-age males. As a result it's entirely possible that our algorithms are tuned to exploit properties of PPG signals that are only found in this demographic.

We thus find our algorithm to be very useful in tracking breath-rate changes in a healthy population, but are unable to draw conclusions about its performance in the case of an opioid overdose. We make recommendations based on the limitations of our methods.

Project Deliverables

List of Deliverables

The original deliverables laid out in early January at the beginning of the project were:

1. An open-source platform on GitHub that can be accessed by everyone. The framework needs to specify the expected input format as well as instructions on how to modify the required configuration settings.
    - Configuration settings will include which models to try with what settings, which evaluation metric to use
    - The framework should be modular and easy-to-use, with the ability for users to add their own models.
        - Models are self-contained functions with clear interface
        - At least some common models from the literature will be implemented
    - The framework should yield performance metrics, visualizations to help inform the user about the strength/weakness of their method(s).
        - Metrics will include common accuracy scores used in literature such as standard deviation and RMSE as well as algorithm execution speed
2. If data is obtained, we were to add and detail the respiratory rate estimation algorithms that work best on this data, evaluating machine learning approaches as well.

However, when data was collected from a real sensor it was clear that there was a very large difference between the data that was publicly available, and data that was manually collected using the max sensor (the latter was much noisier). This led to a shift in the project objectives as we were no longer confident that predicting breathing rate from a ppg signal would be feasible. This led to the outlining of a new set of project deliverables:

1. Collect and provide scripts for collecting breathing rate and PPG data under the following breathing rate situations:
-Normal breathing
-Not breathing
-Slow breathing
-Sudden transition from regular breathing to slow breathing

-Sudden transition from regular breathing to stop breathing

2. Create an algorithm to infer the breathing rate from the PPG signal generated by the Max sensor that works well for tracking the respiratory rate under normal breathing.

3. Quantify the performance of the algorithm on varying conditions such as controlled slow breathing and stop breathing.

4. Create a general framework around this algorithm to allow the sponsor to fit the parameters of the algorithm to attain best performance on newly supplied data.

We were able to accomplish all of the updated deliverables. A total of 32 experiments totaling 263 minutes of data under various circumstances were collected from a group of 5 different people. Multiple features were extracted that were correlated with breathing rate and an algorithm was developed to infer breathing rate from the collected PPG data. The performance of the algorithm and the features were quantified and detailed in this report. Finally, the algorithms were packaged into a framework that will allow the sponsor to train the algorithm on new data once it becomes available.

Financial Summary

The expense for this project is summarised in the following table:

Item	Quantity	Cost
MAX30105 Breakout board.	1	28.56$ CAD

Table 2: Table of project expenses

Ongoing commitments by team members

Until the end of April, the team will continue to provide support to answer any questions on how to use the framework that has been developed.

Recommendations

1. **A larger and more diverse dataset should be collected in order to validate the our results.** The data that we have collected came from a total of 5 healthy college age males. We are not able to make claims about how our algorithms will generalize to the general population, let alone patients undergoing opioid overdoses. As such, we especially recommend getting data from overdose patients so that the performance of existing algorithms can be verified and further algorithms developed.

2. **Further data should be collected with a PPG sensor that can maintain constant finger pressure.** Given the variations in the mean PPG signal that we've observed, we recommend improving the design of the PPG sensor to ensure constant finger pressure. This would eliminate sudden shifts in the mean value of the PPG signal and would allow the use of the

base-wave signal as a feature and could lead to further improvements in the performance of our algorithms.

3. **Analog filtering and amplification circuitry should be considered for the thermistor.** When working the thermistor, noise and discretization effects were clearly visible, with the read thermistor value varying between only 6 discrete values in certain signals. Because we were able to perform filtering on the recovered thermistor signal, this was not a significant issue for estimating the underlying breathing rate. However, we believe that simple amplification circuitry would dramatically improve the quality of the signal recovered from the thermistor.

4. **For the purpose of opioid overdose detection, additional methods to detect overdoses should be considered.** In addition to breathing rate estimation via PPG to detect overdoses, if the Vital Signs team decides to move forward with clinical trials of their hardware on patients under the effects of opioids, we recommend that additional methods of overdose detection be investigated. Namely, the thermistor that was given to us to collect breathing rate was in our experience non-invasive, and very easy to use. If breathing rate estimation from PPG on patients undergoing overdoses proves to be too difficult, we believe that a thermistor based overdose detector would work just as well.

Appendices

Appendix A: Thermistor and Max sensor electrical set-up and data collection code

To collect data from the max sensor and the thermistor an arduino uno was used. The following is a wiring diagram illustrating how everything was connected together:

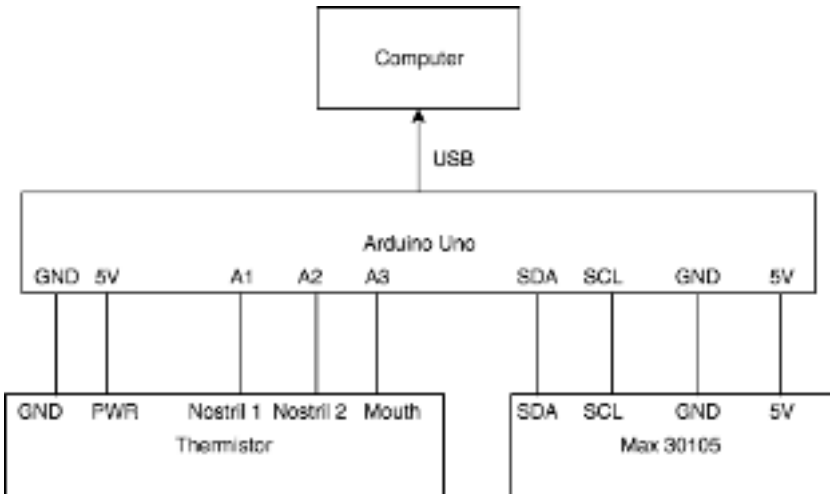


Figure A1: Pinout diagram for the experimental setup.

Code for collecting data can be found at:
https://github.com/fizz-ml/ODetect-Framework/tree/master/collect_data

Appendix B: Framework overview and usage documentation.

This section gives a brief overview of the framework, what it can do, and how to extend it.

Setup

The project was fully developed using the python programming language except for the microcontroller code for collecting data from the max and thermistor sensors. Any version of python 3.5 and above should be compatible. In addition we make use of a few external packages for our project that need to be installed. For this purpose we recommend using conda. The following dependencies need to be met:

1. numpy
2. pytorch
3. scipy

4. pyserial
5. matplotlib

Our framework is divided up into a few submodules. In order for python to be able to cross reference the modules, the root directory of the framework needs to be added to the python path. For Unix systems we have included set_env.sh in the scripts directory. Sourcing this file from the root directory will add it to the python path.
All scripts are designed to be run from the root directory of the frameworkScreenshot from 2018-04-03 20:40:38.

The Feature interface and extending the framework with new algorithms.

The backbone of the framework is the window feature interface. It specifies the functions that any new algorithm needs to obey so that it can be chained together with other features. To implement a new feature, one needs to implement the unimplemented functions and if need be, override the functions that have already been implemented if additional behaviour is desired.

The main functions that need to be implemented by the interface are:
get_param_template(self):
This function returns a template of the parameters that need to be provided to the feature in the form of a python dictionary. The feature should always return the same value and takes no inputs. The implementation of this function for the simple local norm algorithm is shown below.

```
def get_param_template(self):
        param_template = {
            "local_window_length": (float, "The size of the Hanning
window to compute local statistics with in seconds.") # Default was
2000/200
            }
        return param_template
```

calc_feature(self, window):
This function defines the behaviour of the algorithm. It takes in a 1 dimensional numpy array of floating point numbers as inputs and returns another 1 dimensional numpy array as outputs. For example, the CNN takes in a PPG signal as input and returns the predicted mean of the thermistor value.

Optionally, a feature can also inherit from the trainable feature interface. To do so, a feature needs to inherit from the interface and specify a train(training_data,validation_data) function. The train function should use the training data to train the parameters of the feature, and then use the validation data to guard against overfitting.

JSON Model Specification

The Model is specified using a single JSON file. The JSON is a list of objects, each of which represent one feature. Each feature has 4 properties. A name, which is a string.
The in_features, which is a list of integers corresponding to the features that the current feature takes as input, the type, which is any of a specific set of strings and specifies the corresponding features type and the params which are the fixed parameters associated with a feature.

Of note, the integers in the list of integers reference features starting at zero, so the first feature in the list is feature 0, the second is feature 1, etc. In addition the allowed input features are only the preceding features, so for feature 5 the only acceptable input features are features 0-4.

The set of valid type strings and their corresponding features are.
-"id" : IdentityWindowFeature
-"ea" : WindowEnvelopesAmplitude
-"ptp" : WindowPeakTroughPeriods
-"butter" : SimpleButterFilter
-"spline" : SimpleSplineFilter
-"local_norm" : SimpleLocalNorm
-"breath_cnn" : BreathCNNFeature

When adding a new feature, a new entry in the dictionary in model.py under the utils directory should be added so that it can be referenced in a JSON.

An example JSON is provided for our current best algorithm:

```
[
    {
        "name": "input_signal",
        "type": "id",
        "in_features": [],
        "params": {}
    },
    {
        "type": "norm",
        "in_features": [0],
        "params": {}
    },
    {
        "type": "butter",
        "in_features": [1],
        "params": {"low_cut": 0.05, "high_cut": 1.5, "order": 3}
    },
```

```
    {
        "name": "filtered_input",
        "type": "local_norm",
        "in_features": [2],
        "params": {"local_window_length": 40}
    },
    {
        "name": "EA",
        "type": "ea",
        "in_features": [3],
        "params": {"lookahead_length": 0.025, "delta": 0.02,
"normalize": true}
    },
    {
        "name": "PP",
        "type": "ptp",
        "in_features": [3],
        "params": {"lookahead_length": 0.025, "delta": 0.02, "interp":
"spline", "s": 0, "toggle_p_t": true, "normalize": true}
    },
    {
        "name": "PT",
        "type": "ptp",
        "in_features": [3],
        "params": {"lookahead_length": 0.025, "delta": 0.02, "interp":
"spline", "s": 0, "toggle_p_t": false, "normalize": true}
    },
    {
        "name": "CNN",
        "type": "breath_cnn",
        "in_features": [3, 4, 5, 6],
        "params": {
            "channel_count" : 15,
            "dilation" : 10,
            "field" : 11,
            "layers" : 5,
            "down_sample" : 12,
            "train_time" : 750,
            "learning_rate" : 0.0002,
            "param_path" : "models/17.t7"
        }
    }
]
```

Building a model from a JSON and data.

After the Json has been specified, all that needs to be done to construct the model using the train_model.py script found in the train model directory. The script take in 4 inputs,
-The sampling rate
-The json specification of the model to train
-A folder containing hdf5 files for training
-A folder containing hdf5 files for validation

To construct the hdf5 files, the format_max.py script can convert the raw CSV files with the following columns into appropriately formatted hdf5 files. The CSVs need to have the following columns in the following order:
-Timestamp
-PPG value
-Nostril 1 thermistor
-Nostril 2 thermistor
-Mouth thermistor

An example invocation of the model building script is shown below:

```
python scripts/train_model.py 200 models/26_alone.json
data/data_sets/example/train/ data/data_sets/example/val
```

Visualizations

After a model has been trained, several scripts can be run in order to visualize the output of the model. All of these scripts generally take an input data file path and the path to the model description json file. All visualization scripts can be found under the exploration folder.

The available visualization scripts are:

visualize_model_features.py
Visualizes each feature in the model in the order specified in the model description. Plots the features against the target signal. This is mainly used to debug a large model by taking a look at each of its feature components.

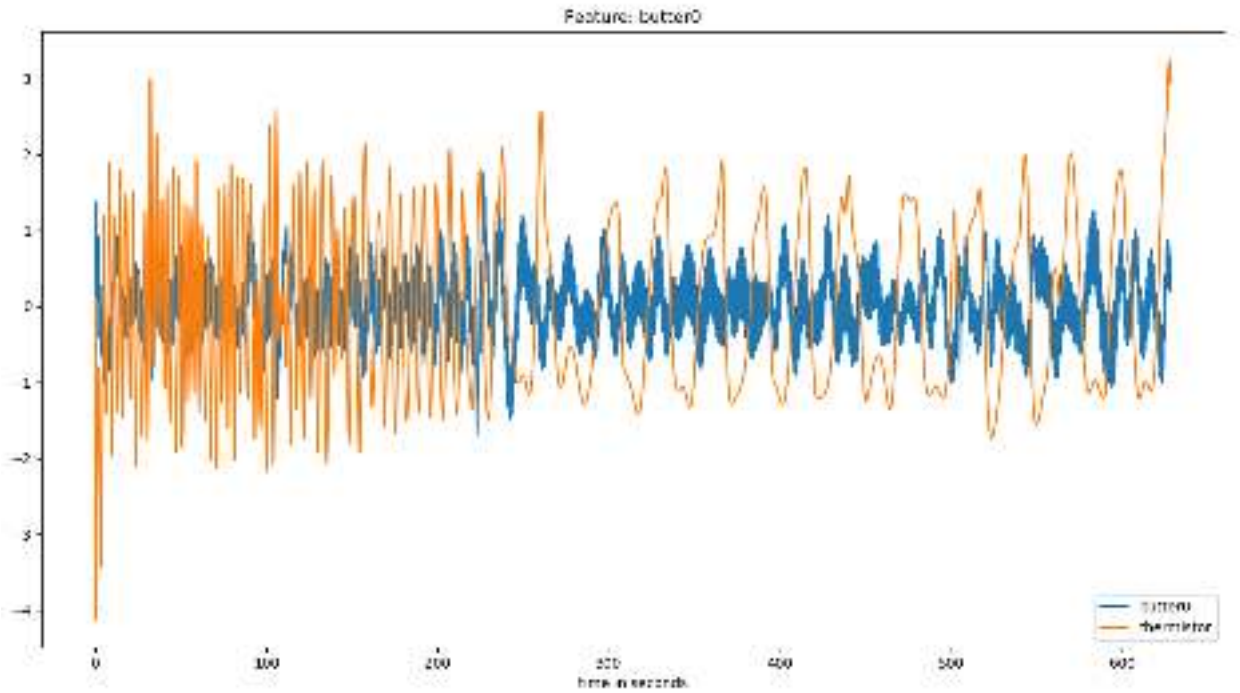


Figure B1: Example of running visualize_model_features on the CNN model. It depicts butter0 which is the automatically generated name for the SimpleButterFilter feature that is used by the CNN model.

show_stft.py
Visualization of the STFT plot of a given model. Computes the short time fourier transform of the output of the model and plots it as a 2D color meshgrid plot. This is mainly used to discover if a feature has a clear frequency profile over time and whether it could correlate to the breathing rate.

show_compare.py
Visualizes the immediate outputs and the instantaneous RR prediction for a list of models given as input along with the target thermistor signal given. This is mainly used to compare two seperate features and their correlation to the target thermistor signal.

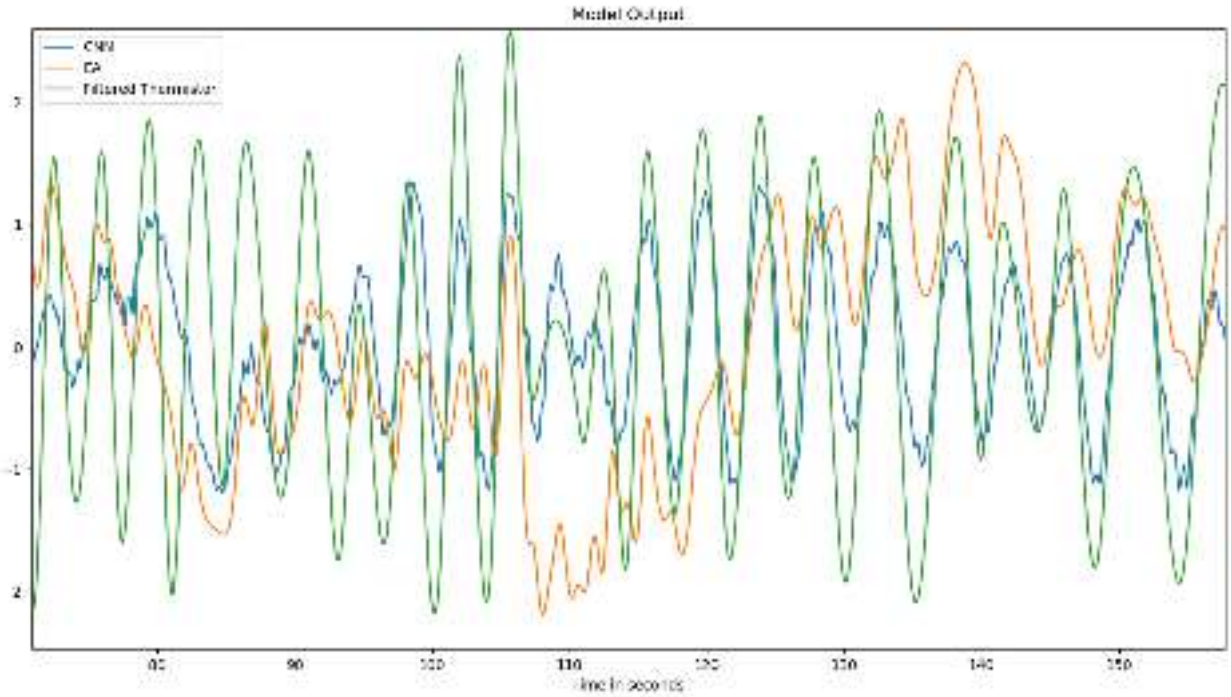


Figure B2: Plot generated by show_compare which compares the output of the EA feature against CNN and the filtered thermistor on experiment 28.

compute_error.py

Plots of predicted centroid and the actual centroid and predicted instantaneous breathing rate and the actual instantaneous breathing rate. This script is mainly used to visualize the output prediction of a given model. It also computes the root mean squared error.

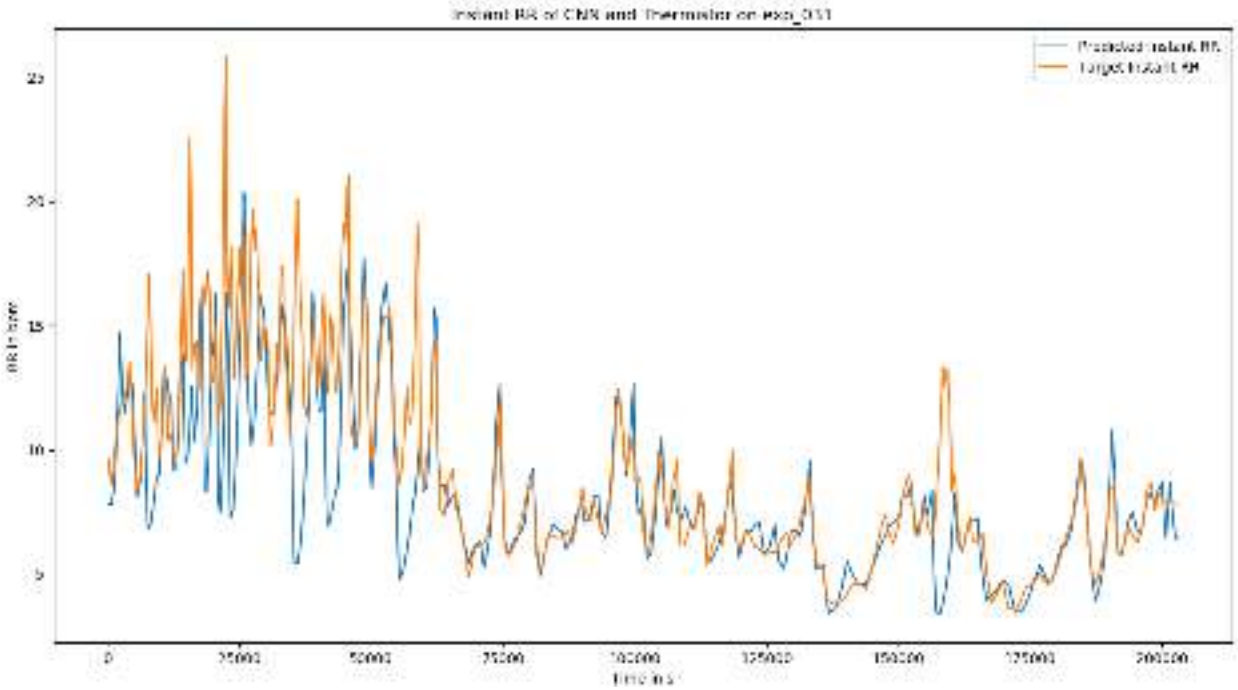


Figure B3: Plot of the predicted instant respiratory rate and actual respiratory rate generated using compute_error on Experiment 31 and the CNN model.

Appendix C: Experiment Log

Experiment Logbook, Team 1807, Engineering Physics 459

Experiments 001 through 004 all used a sampling rate of 100Hz on the max sensor
09/03/18 5:35 PM

<u>Experiment 001</u>
Goal: Comparing the Masimo against the max sensor.

Subject: Anmol

Experiment_001_max: Max sensor
Experiment_001_masimo: Masimo sensor

- Right hand, index finger in masimo, middle finger on Max

- Battery level of Masimo was red.
- Logging the pleth signal generated from both at the same time on 2 machines
- Windows machine running android studio masimo client sample code and using adb logcat to pip to csv file
- Linux machine running arduino script with arduino uno and python log_masimo.py script to log data from the max sensor at baud rate of 115200 and 200Hz sampling rate
- Subject was Anmol, talking intermittently, mostly at rest

- Anmol deliberately lowered his heart around unix time ...45910.(...) 7 minute mark noise from drilling in the background possibly stressing out subject

Subject started laughing 46000

<u>Experiment 002</u>

Subject: Anmol

- Anmol will not react to the readings on the device
- Middle finger in Masimo
- Index finger on max
- Similar setup to experiment 001 otherwise

Experiment 003
Subject: Anmol
Left hand max
Right hand masimo
Start: 03-09 18:20:48.572 D/PPG

Using index fingers
Calm normal breathing, not looking at device numbers. Occasional talking.

Experiment 004
Subject: Anmol
Left hand Masimo
Right hand max

Normal breathing start

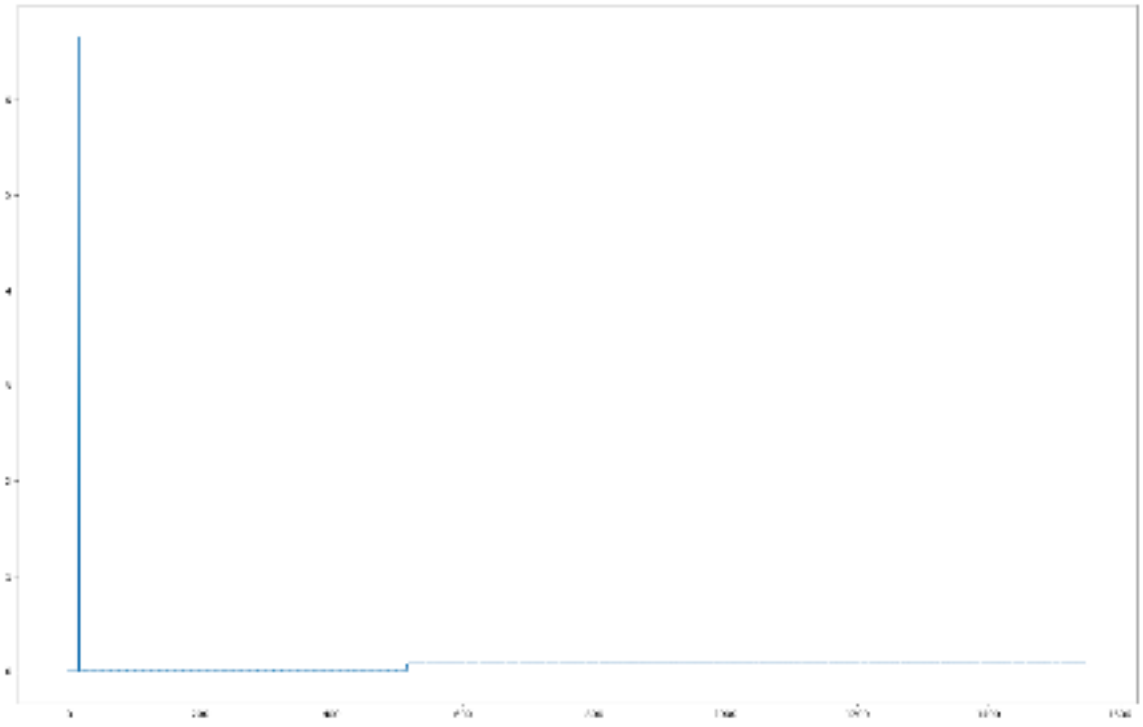
At 1:30 stop breathing
Start again at 2:00
Stop 3:30 and start 4:00
Repeating every 2 minutes

Curtis counting down breathing
Anmol raised his arm on second normal breath period
Counting down only 5 seconds before transitions
Raised arm on third normal breath period
Anmol amused on stop of period 4
Curtis tells story
Period 5 stop breath

Other Notes:

Found that the sampling rate can vary between samples quite a bit for 100Hz setting, time between samples varied from 78ms to 82ms on the max.
Using 400 Hz setting, time between samples had major variation between 0.0002 and 0.003 ;seconds. -> 0.2ms and 3 ms
This time is logged in python and so buffering might be causing this.
Plotting the running average sampling rate showed a step up at 500 samples.

<u>Experiment 005</u>
Anmol right index finger
On 400Hz we recorded 2 minutes of data
After removing the print statement
2000 samples were recorded and the exact error was observed (almost identical)



Plot of time interval between samples

- Serial logging mircos on arduino
- When looking at the time intervals, still has spike at 500 samples
- In addition micros stops increasing at a certain value and remains cosntant
- On sample number 529 we found that millis recovers and is normal from then on out
  T2.csv recorded on Anmol's laptop

11/03/18 1:30 PM

Time average time between samples as a function of the sampling frequency passed to the Max sensor.

800Hz	0.16
400Hz	0.08
200Hz	0.04
100Hz	0.08
50Hz	0.16
1000Hz	0.04
1600Hz	0.16
3200Hz	0.08

Removed writing to disk in inner loop python script
Rerunning with change

3200Hz	0.08

Ruling out bottleneck on python side

Tried changing pwm to 69 (all previous were at 411)
Didn't change anything

Pwm back to 411, trying sampleAverage=1 all previous at 8

3200Hz	0.01
200Hz	0.006

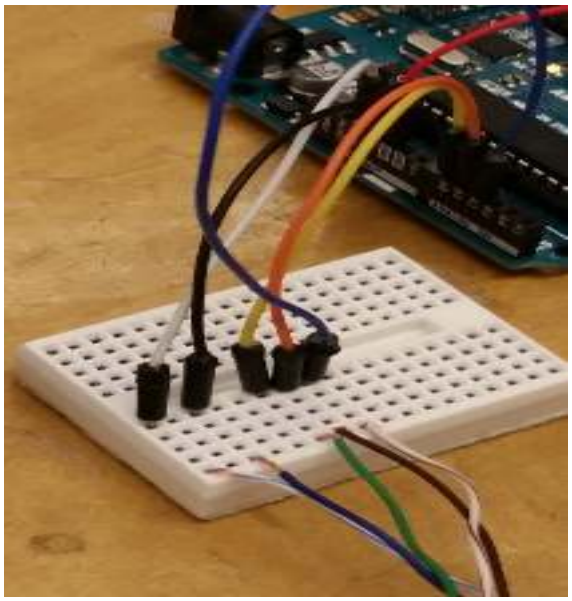
It's just 8 times faster (meaning original samplerate is till not being respected), but now 200Hz is actually about 200Hz

By using a rubber we can significantly reduce offset variations by stabilizing the pressure between finger and max

Experiment 006

Curtis right hand index finger on max and thermistor worn. Talking throughout
200Hz sampleAverage=1

Wiring of thermistor:

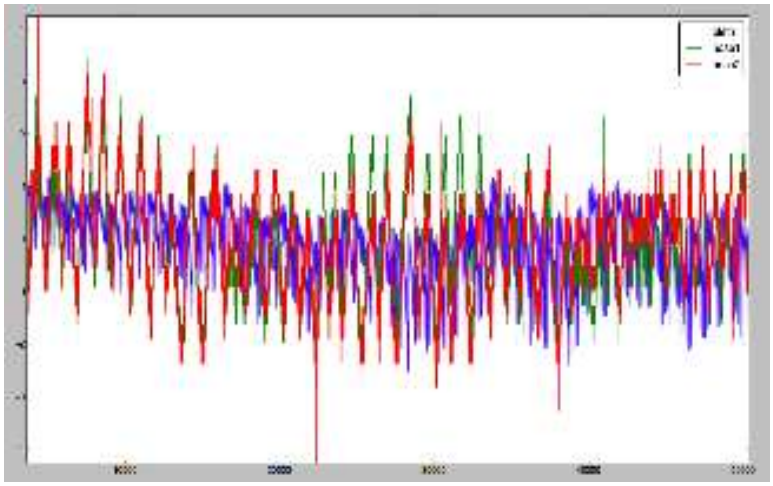


Code for thermistor:

```
void loop()
{
  Serial.print(particleSensor.getIR()); //Send raw data to plotter
  Serial.print(", ");
  Serial.print(analogRead(A0)); //Send raw data to plotter
  Serial.print(", ");
  Serial.print(analogRead(A1)); //Send raw data to plotter
  Serial.print(", ");
  Serial.println(analogRead(A2)); //Send raw data to plotter
}
```

interpolate_spline

Experiment 007:
Curtis right hand index finger on max and thermistor worn. No talking. Otherwise identical
200Hz sampleAverage=1



Plot of normalized plethysmogram signal(blue), nostril 1 (green) nostril 2 (red)

Experiment 008:
Curtis right hand index finger on max and thermistor worn. No talking. Holding breath.

Otherwise identical.
200Hz sampleAverage=1

Experiment 009:
Masimo low on power (red)
Curtis right hand index finger on max, right hand middle finger in masimo and thermistor worn. No talking. Normal breath. Otherwise identical.
200Hz sampleAverage=1

Logging command: ~/Projects/459/mightysat_client_sample# adb logcat -v "time" -s "PPG" | ts "%.s" > ../experiments/experiment009_masimo.txt

Experiment 010:
Same experimental setup as 009, but using the stop start breathing cycle as in experiment 004
Experiment prematurely ended with Curtis laughing

Experiment 011:

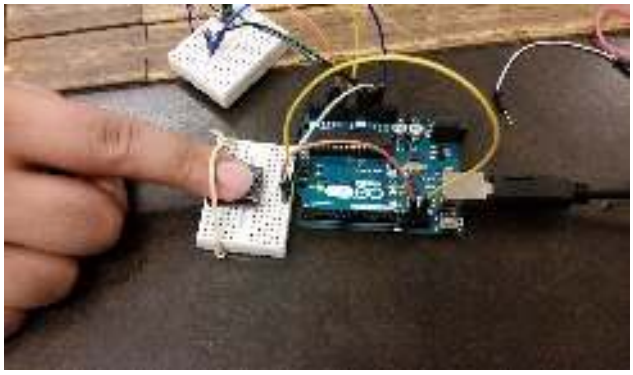
Rubber band on the Max has side effects. Causes finger indentation as seen above. Photo taken right after conclusion of experiment 10

17/03/18
Experiment 12:
Curtis wants more break-time to avoid suffocating
Slowing down of breathing rate: 1 minute normal breathing then 45 seconds slowed down. No talking from Curtis.

Experiment 13:
Curtis 1 min hyperventilating- 1 min normal breathing rate for total 10 mins. Same conditions as before.. Curtis also reading a book on his laptop

Experiment 14:

Set-up as follows. 10 minutes of Anmol breathing. Anmol talking for the first 30 seconds, otherwise quiet and composed.



Experiment 15

10 minutes of Cheng breathing. Otherwise identical conditions.
Note: many features of this signal are erratic when compared to other experiments. Experiments carried out on cheng's computer, precision on unix timestamps is significantly lower.

Experiment 16

10 minutes of Cheng breathing. 5 Cycles of 1:30 regular breathing followed by 30s regular breathing. Otherwise identical conditions.

Experiment 17

10 minutes of Tim Branch breathing.
Regular breathing. Otherwise identical conditions.

Experiment 18

10 minutes Cheng slow breathing. Otherwise identical conditions.

25/03/18

Experiment 19

Curtis breathing trying to breath slowly, left nostril partially obstructed. From 2:32 to 2:53

26/03/18

Experiment 20

Cheng trying to breath at a slow pace. Experiment prematurely ended because wire plugged out, about 20-30s of data we're collected. Do not use.

Experiment 21

Cheng trying to sleep.

Experiment 22

Anmol trying to breath slowly.

Experiment 23

Curtis *really trying* to breath slowly. Seems to be struggling

Experiment 24
Anmol breathing slowly but more consciously.

27/03/18
Note: experiments 25 and beyond are carried out using a new Max 30105 sensor.

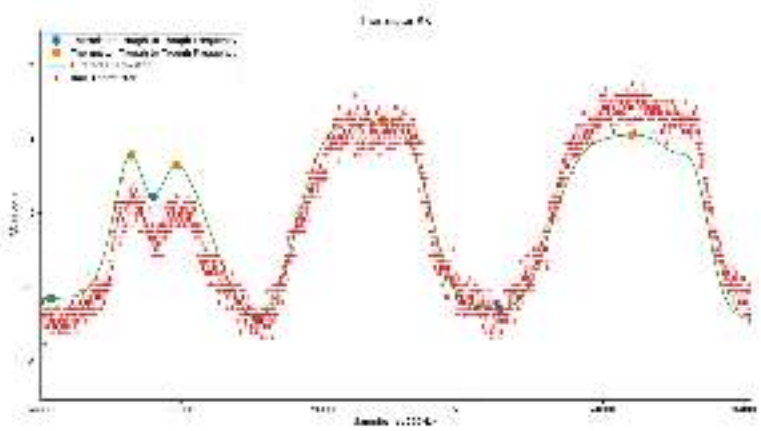
Experiment 25
Curtis breathing normally for 1 minute followed by slow breathing for 10 minutes.

Experiment 26
Subject: Rahat
Breathing normally for 5 minutes followed by breathing slowly. He was also simultaneously reading from his laptop.

Experiment 27
Subject: Rahat
Breathing normally for 10 minutes. Same condition as before

Experiment 28
Subject: Cheng
Cheng breathing normally
Experiment 29
Subject: Cheng
Cheng breathing normally

29/3/18
After discovering some inconsistencies in the Butterworth filter for filtering the breathing rate, started using SimpleSplineFiltering



Example of ambiguous case. It is not exactly obvious where the exact peak or trough should be in the original signal as they are quite broad.

30/3/18

Experiment 30

Curtis Slow breathing for 15 minutes.

Experiment 31

Curtis 5mins regular breathing followed by slow breathing

Experiment 32

Anmol trying to sleep. Not completely sleeping nor completely awake. Seemed like a short nap. At around t = 1522451400 posture was changed from sitting to laying back in order to facilitate sleeping.

References

1. Malamed, S.F. (2007). Medical Emergencies in the Dental Office - E-Book. Elsevier Health Sciences
2. Administration of INTRAMUSCULAR Naloxone for Suspected or Confirmed Opioid Overdose, BC Emergency Health Services (http://www.bcehs.ca/about-site/Documents/Naloxone%20reference%20guide%20for%20first%20responders.pdf)
3. Johansson, Anders. "Neural network for photoplethysmographic respiratory rate monitoring." Medical and Biological Engineering and Computing 41.3 (2003): 242-248.
4. http://www.capnobase.org/database/pulse-oximeter-ieee-tbme-benchmark/
5. http://cs231n.github.io/neural-networks-3
6. http://iopscience.iop.org/article/10.1088/0967-3334/37/4/610