2020

Utilizing Machine Learning For Respiratory Rate Detection Via Radar Sensor

Anwar Elhadad
University of Vermont

UTILIZING MACHINE LEARNING FOR RESPIRATORY RATE DETECTION VIA RADAR SENSOR

A Thesis Presented

by

Anwar Khalil Elhadad

to

The Faculty of the Graduate College

of

The University of Vermont

In Partial Fulfillment of the Requirements
for the Degree of Master of Science
Specializing in Electrical Engineering

January, 2020

Defense Date: November 11, 2019
Thesis Examination Committee:

Tian Xia, Ph.D., Advisor
Safwan Wshah, Ph.D., Chairperson
Luis Duffaut Espinosa, Ph.D.
Cynthia J. Forehand, Ph.D., Dean of the Graduate College

ABSTRACT

In this research, we investigate a data processing method to capture the respiratory rate of a person by utilizing a doppler radar to monitor their body movement during respiration. We utilize a machine learning algorithm with a radar sensor to capture the chest movement of a person while breathing and determine the respiratory rate according to that movement. We are using a Random Forest classifier to distinguish between different classes of pulses. After that, the algorithm constructs a sinusoidal signal representing the breathing rate of the sample. By applying this technique, we can detect the breathing rate accurately for different subjects by analyzing the evolution of the reflected pulse while breathing. Furthermore, we can detect the change in pulse width ratio between the pulses of the classes across multiple breaths.

ACKNOWLEDGEMENTS

People to acknowledge:

1 - Tian Xia

Dr. Xia was a great help as an advisor and a source of information and feedback.

2- Safwan Wshah

Dr. Wshah experience in the field was helpful and very much appreciated.

3- Luis D. Espinosa

Dr. Espinosa was always available to help, even on short notice.

4- Tim Sullivan

Timothy Sullivan was a great research partner to whom I owe the success of this research.

5- Jennifer Nigro

Despite her unfamiliarity with the topic, she was always available to help with writing tips and feedback.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1: INTRODUCTION

There are many physiological parameters to evaluate a patient’s health conditions, such as respiratory rate, heart rate, blood pressure, and sugar levels. The monitoring of these parameters can be challenging, especially in cases of Intensive Care Unit patients who need to be monitored at all times. Typically, to monitor the changes in health status, several tools and machines connected to the subject are utilized to measure these variables in real-time. The pads and wires used for monitoring restrict patient movement and impair comfort. In extreme cases, the constant contact of the pads and wires with a patient's skin can cause swelling or contact dermatitis. Therefore, performing vital signs monitoring wirelessly is of great value.

In the literature, there are many pieces of research exploring ultra-wideband (UWB) remote sensing for health care applications. In [1], the radar system is utilized to allow the signal from the transmitting antenna (Tx) to penetrate the human subject's body along its path to the receiving antenna (Rx). In the setup, the test subject stands between the Tx and Rx which is generally inconvenient for mounting the radar system. In practical operations, a more efficient setup is to place the Tx and Rx antennas on the same side of the test object which can provide more operating flexibility. In many other pieces of research, their focuses are on the frequency domain analysis of the radar signals for measuring respiratory signals, such as Fourier transform [2], short-time Fourier transform (STFT) [3] or Hilbert-Huang transform (HHT) [4]. In [5], an adaptive digitized algorithm is developed to precisely detect the movement of a robotic arm, heartbeat, and respiration activities by utilizing a contactless pulse-based UWB radar sensor. In [6], an approach to identify and track the reflection from a breathing subject uses an autocorrelation coefficient on pairs of frames. A recent approach, shown in [7], uses an autocorrelation-based variational mode decomposition algorithm technique to measure the changes in HR and RR signals. In [8], researchers created a demo to track whether a person is holding their breath or not in real-time using wireless signals. In [10], A microwave technique for measuring respiratory movements of man and animal is described. The technique is noncontracting and is based on the scattering of continuous-wave radiation. In [11], a non-contact system was set up to perform overnight monitoring of vital signs from different directions using low power radio waves. In [12], A radar is used to measure the respiration pattern that is used to infer the tumor location in real-time.

One of the functional current approaches is a real-time system that can monitor an infant's respiration and detect apnea when it occurs. A location algorithm is applied periodically to track the current location of the infant's chest. However, this approach is still under study as capturing the rate for a long period of time is challenging due to the obstruction caused by the infants’ movement.

The major limitation with existing approaches is their inability to distinguish between the periodic chest movement of a human subject during the breathing and other periodic movements of a nonhuman object. Human subjects' chest movement in breathing alters the pulse shapes of radar reflection signals. Moreover, the shape of the reflected pulse is in continuous evolution since the chest thickness changes between inhaling and exhaling states. As a result, the radar reflection signal's pulse width and amplitude vary depending on the radar cross-section view of the chest area. Moreover, in the test environment, there might exist other periodically moving objects that carry frequencies close to that of the chest movement which can cause interference to degrade the measurement accuracy. The direct frequency analysis approaches [2-8] can hardly distinguish between the periodic movement of the breathing chest and other periodically moving objects since they assume a fixed shape of the radar reflection pulse for both scenarios.

In this paper, we develop a new approach to precisely detect respiratory rate (RR) by utilizing a machine learning (ML) approach to analyze the signals from a UWB radar sensor. By using a commercialized high-quality radar system, we are able to observe the de-modulated received reflection pulse signal. Subsequently, we are able to train the ML algorithm model to distinguish different forms of the received pulse when being reflected from a human subject, where the pulse shapes resulting from the respiration are classified into two categories. The first represents the pulse shape when the lungs are filled with air corresponding to inhalation, and the second represents the pulse shape corresponding to exhalation. The ML algorithm is trained to recognize the different shapes of the pulse during its evolution from exhalation to inhalation and vice versa, which enables characterizing pulse shapes, rather than solely detecting the periodic movement of the reflected pulse. As a result, it facilitates to avoid misidentifying periodic movement that resembles the respiratory. In addition, using this technology, we are able to further examine respiratory behavior by analyzing the sensing signal’s inhalation and exhalation features.

CHAPTER 2: EXPERIMENTAL BASIS

2.1. Radar

We are using XeThru X4M200 is a complete IR-UWB radar system on a chip. To configure it correctly, it's important to have a good understanding of how an impulse radar system works and how the received data is sampled and presented. The fundamentals of an impulse radar system are shown in Fig. 1. The radar sends out an electromagnetic impulse through the Tx antenna which is reflected from any object in front of it. The reflections travel back and are received and sampled through the Rx antenna.

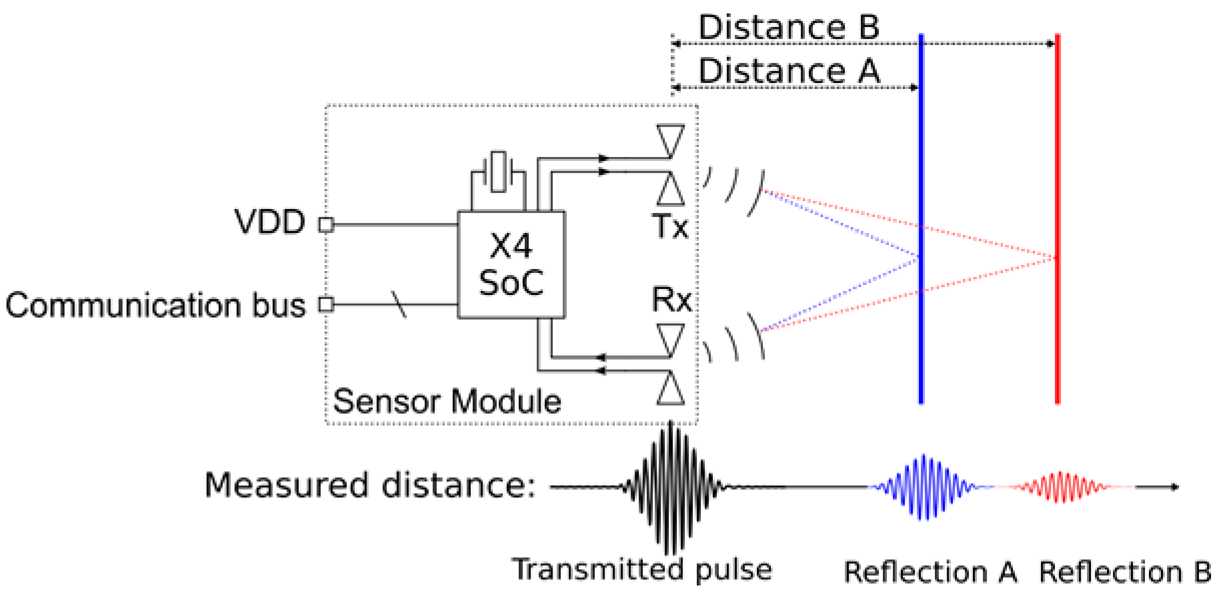


Fig. 1. Basic UWB radar concept.

The pulse that X4M200 transmits is configurable within two different bands, supporting worldwide regulations. The lower pulse generator setting enables transmission within the band 6-8.5 GHz, shown in Fig. 2-A, and the high settings within the band 7.25-

10.2 GHz. To receive the reflected energy, it uses a high-speed sampler with a sampling rate of 23.328 GS/s that can sample up to 1536 samples. Since electromagnetic waves travel at the speed of light, this corresponds to a sampling of reflected pulses in a window of about 9.9 meters long. Each sampling point is referred to as a range bin.

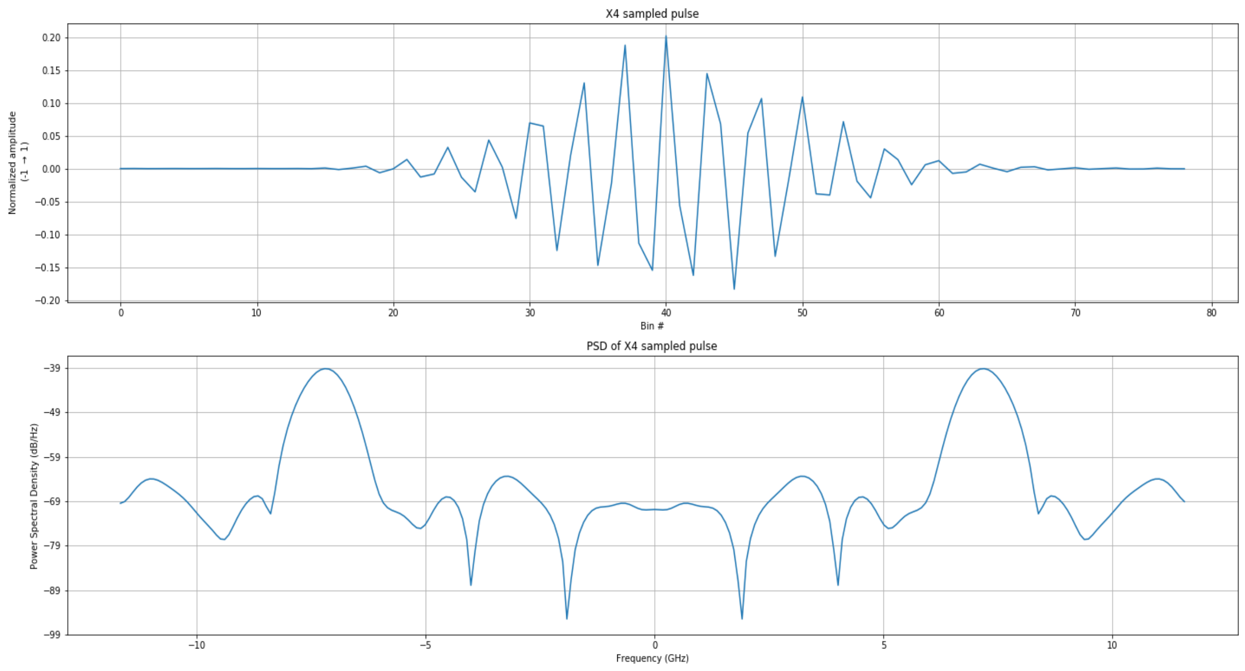


Fig. 2-A. Transmitted pulse by X4M200.

When the pulse is transmitted from the radar, it travels in cone form until it reflects from surfaces of reflections. While the pulse is traveling the signal loses its energy by the increase in the distance traveled causing the pulse to lose some of its amplitude. Fig. 2-B illustrates the energy loss of a reflected pulse based on its distance.

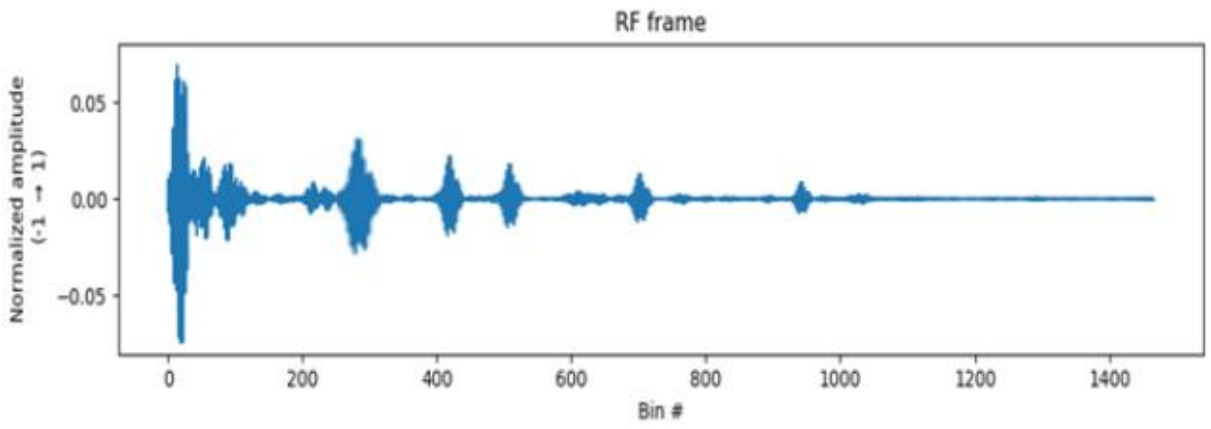


Fig. 2-B. Energy loss of a reflected pulse.

In Fig. 2-C, we can see a drop it terms of energy pulse the further we move away from the radar. A reflection energy depends on the size of the reflecting surface and the distance of that object from the radar. We used the same object (Aluminum sheet) to generate the reflection at different distance.

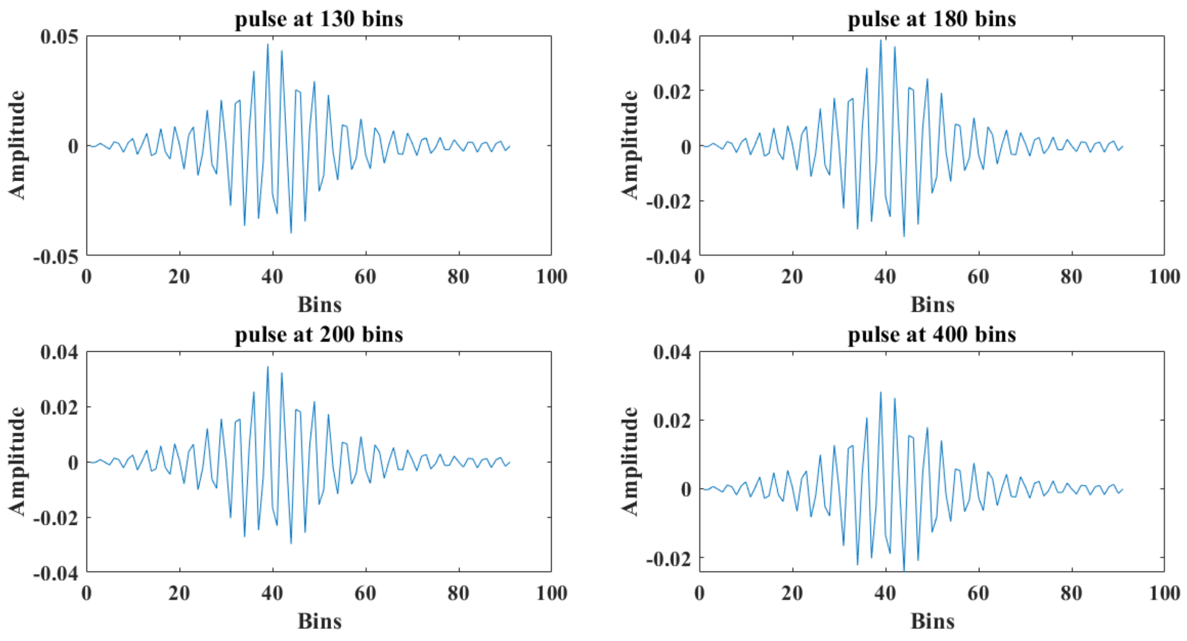


Fig. 2-C. Pulse reflection at different distances.

In Fig. 3-A, we can see the result from sampling data over the 1536 bins, referred to as a radar frame. By starting the sampler right after transmitting a pulse, an object 2 meters away shows up on the frame as a pulse around range bin 280. The energy seen from bin 0 is caused by the energy transmitted directly from Tx to the Rx antenna.

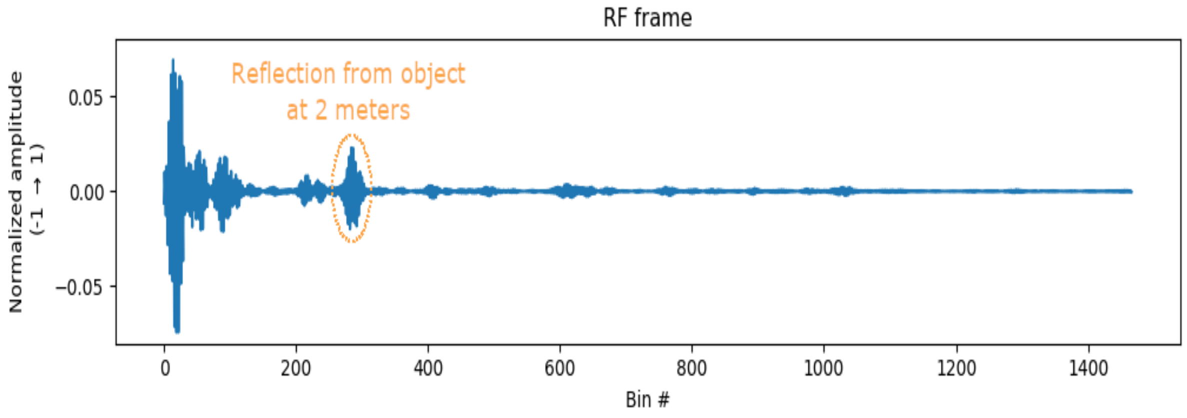


Fig. 3-A. Radar frame.

The radar captures the reflections of every object in its field of view, and the distance to the object corresponds to the position of the reflection in the radar frame. Larger objects give reflections with higher amplitudes and the amplitude of the reflection is reduced the further away

the object is. The object material also plays a role as different materials give different reflections,

for example, a metal object gives a larger reflection than a plastic object. The amount of energy reflected on the radar is referred to as the object's radar cross-section (RCS).

To measure the displacement using the radar we are utilizing the Doppler effect. The Doppler effect causes the received frequency of a source (how it is perceived when it gets to its destination) to differ from the sent frequency if there is a motion that is increasing or decreasing the distance between the source and the receiver. This effect is readily observable as variation in the pitch of sound between a moving source and a stationary observer.

When the distance between the source and receiver of electromagnetic waves remains constant, the frequency waves is the same in both places. When the distance between the source and receiver of electromagnetic waves is increasing, the frequency of the received waveforms is lower than the frequency of the source waveform. When the distance is decreasing, the frequency of the received waveform will be higher than the source waveform.

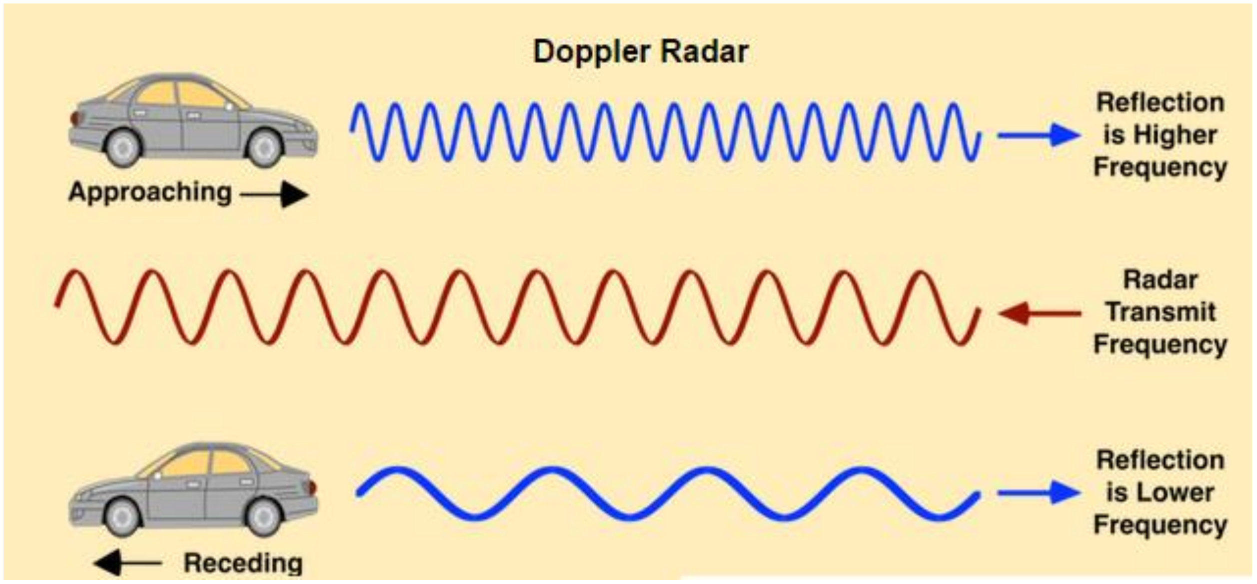


Fig. 3-B. Doppler effect.

2.2. Analysis

In this study, an impulse ultra-wideband radar kit X4M200 (Fig. 4-A) is utilized as the sensing device. The operating mechanisms of the UWB radar for measuring the respiratory are shown in Fig. 4-B, where the electromagnetic pulses are radiated through the transmitting antenna Tx. The signal reflected from the objects travels back to the radar receiver.

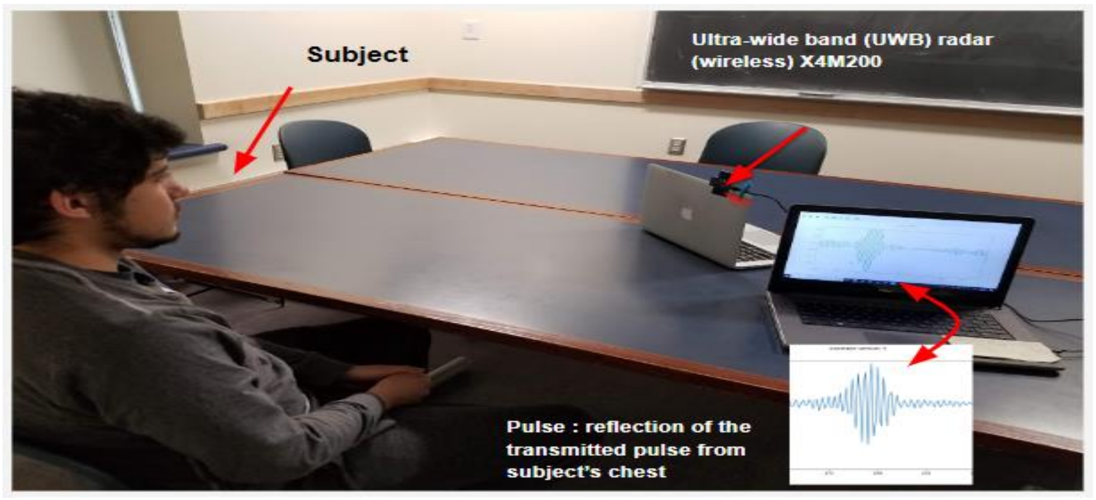

Subject
Ultra-wide band (UWB) radar (wireless) X4M200
Pulse : reflection of the transmitted pulse from subject's chest

Fig. 4-A. X4M200 radar sensor (setup with sensor).

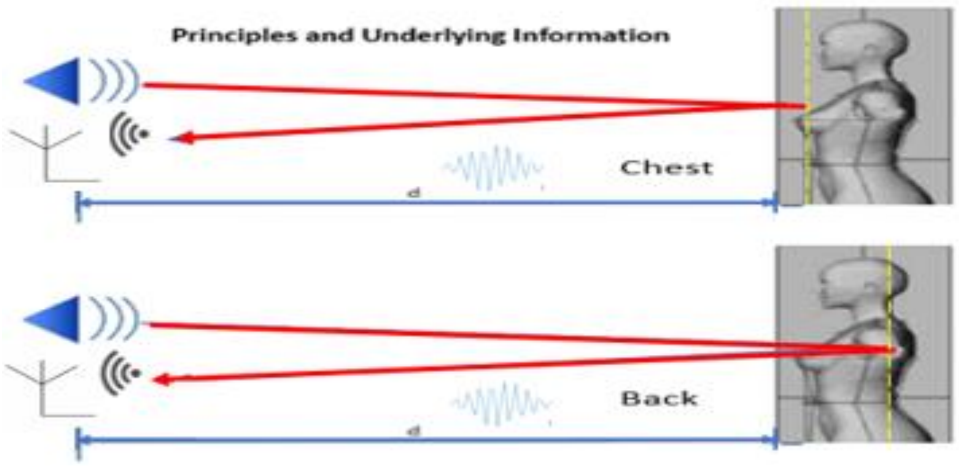


Fig. 4-B. Basic radar function.

For the respiration detection, the reflection signals are mainly produced from two parts of the human body, one is from the front surface of the chest, and the other is from the back surface of the body. The distance between these two bodies surfaces changes during inhalation and exhalation. As a result, it causes variations of the reflection signal, including the pulse width and pulse amplitude.

To detect an object at a distance (Range), the wave packet travels a round trip distance of $2 \times Range$. The range from the radar to the object is calculated based on Eq. 1, where C is the speed of light, and τ is the signal travel time, and factor 2 accounts for the round trip.

$$Range \ = \frac{C \times \tau}{2} \qquad (1)$$

After the pulse is transmitted, the Rx receives the reflected signal from the subject. The distance between the antenna and the subject’s chest can be derived as:

$$C(t) = d0 + dr1\, sin(2\pi f t) \qquad (2)$$

where d0 represents the distance between the antenna and human chest front surface, dr1 is the displacement amplitude chest, and f represents the respiration frequency. Similarly, the distance from the subject's back surface can be derived as:

$$B(t) = (d0 + db) - dr2\, sin(2\pi f t) \qquad (3)$$

where dB represents the torso thickness which is the distance between the chest surface and the back surface and dr2 is the displacement amplitude of the back surface movement.

The negative sign indicates that the back-surface moves in the opposite direction in regard to the chest front surface during respiration.

As a person breathes, the torso surfaces move back and forth periodically. As a result, the radar reflection pulse from the chest surface arrives at the radar receiver progressively sooner while inhaling air. During exhalation, the chest moves away from the detector. Thus, the reflected pulse arrives progressively later while exhaling (Fig. 4-C).

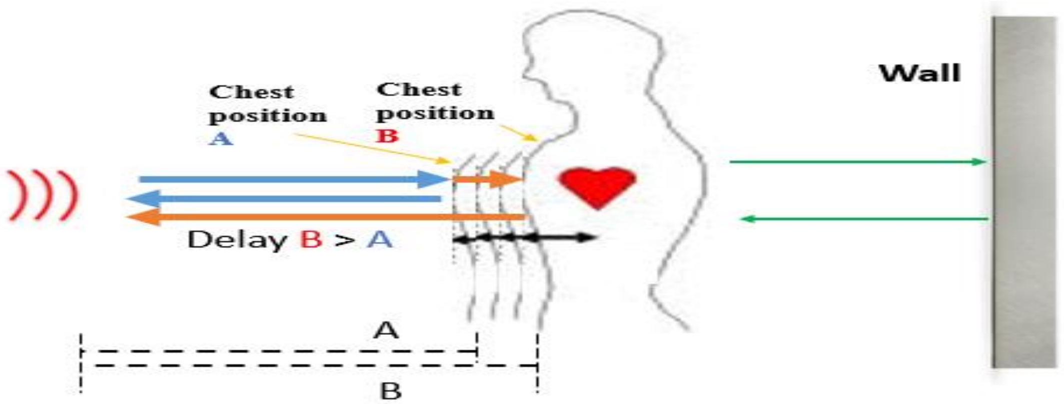


Fig. 4-C. Respiratory rate measurement mechanism.

Generally, the reflection pulse from a fixed shape moving object maintains its form. The only change occurs to the amplitude of the pulse which increases when the object moves towards the radar and decreases when the object moves away. In Fig. 4-D, we have a reflection sample from a rectangular aluminum sheet that retains its width and envelope even when moving the sheet. The aluminum sheet is 2 square feet in size and is placed 1.2m away from the radar as illustrated in Fig. 5-C. This test is conducted to show how the reflected pulse is different when reflecting from a fixed shape object versus reflecting from a human subject. Fig. 5-A shows the periodically moving aluminum sheet reflection waveform maintains its shape. The only change is the amplitude. However, for the moving chest, the width of the reflected pulse increases during inhalation and decreases during exhalation, such evolution is illustrated in Fig. 5-B.

In Fig. 5-D, Pearson correlation, as illustrated in equation 4, is used to validate that the pulse reflected from the aluminum sheet maintains its shape. The correlation does not drop below 0.9 across all frames, indicating that the pulse retains its shapes in every frame.

$$r_{xy} = \frac{\sum_{i=1}^{n}(x_i - \underline{x})(y_i - \underline{y})}{\sqrt{\sum_{i=1}^{n}(x_i - \underline{x})^2}\sqrt{\sum_{i=1}^{n}(y_i - \underline{y})^2}} \qquad (4)$$

where r_{xy} is the Pearson correlation coefficient, *n* is the number of sample points, and *x(i)* and *y(i)* are the individual sample points indexed with *i* in different pulses.

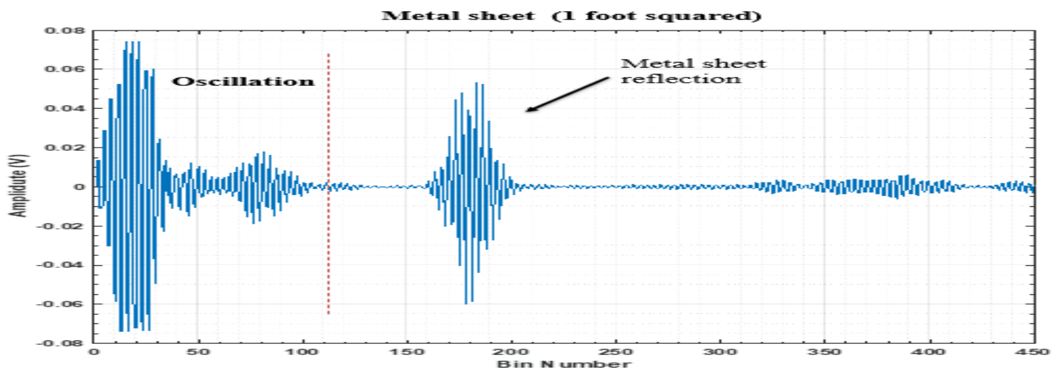


Fig. 4-D. The reflected pulse from a metal sheet (1 square foot).

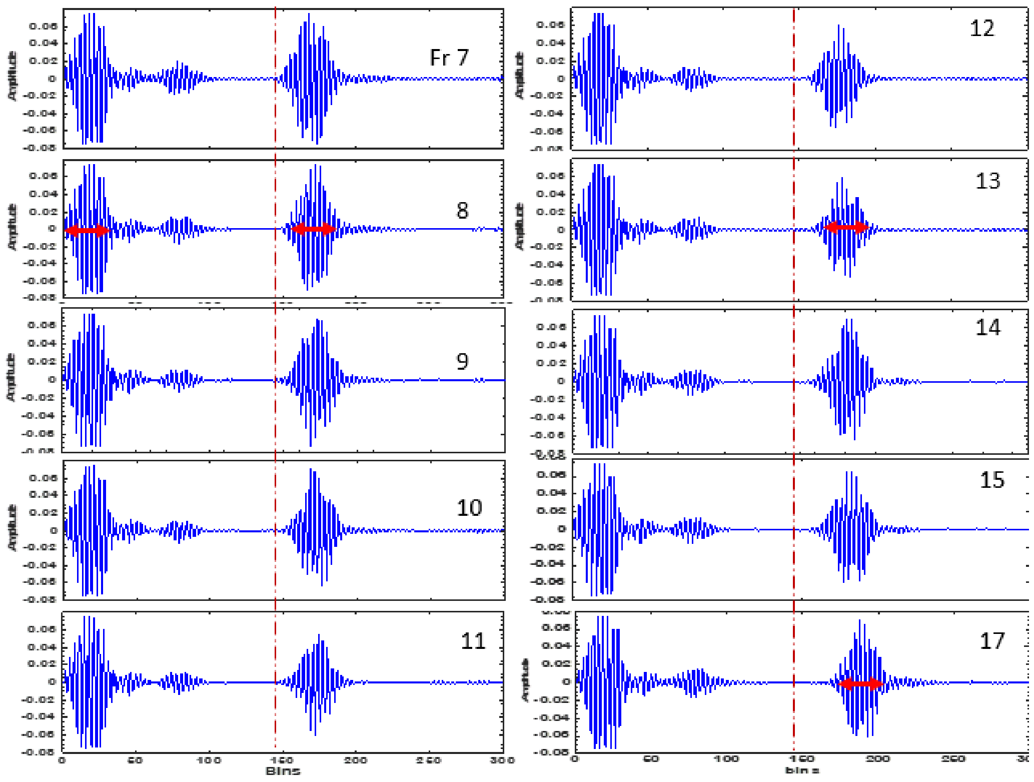


Fig. 5-A. Reflected pulse evolution of an aluminum sheet.

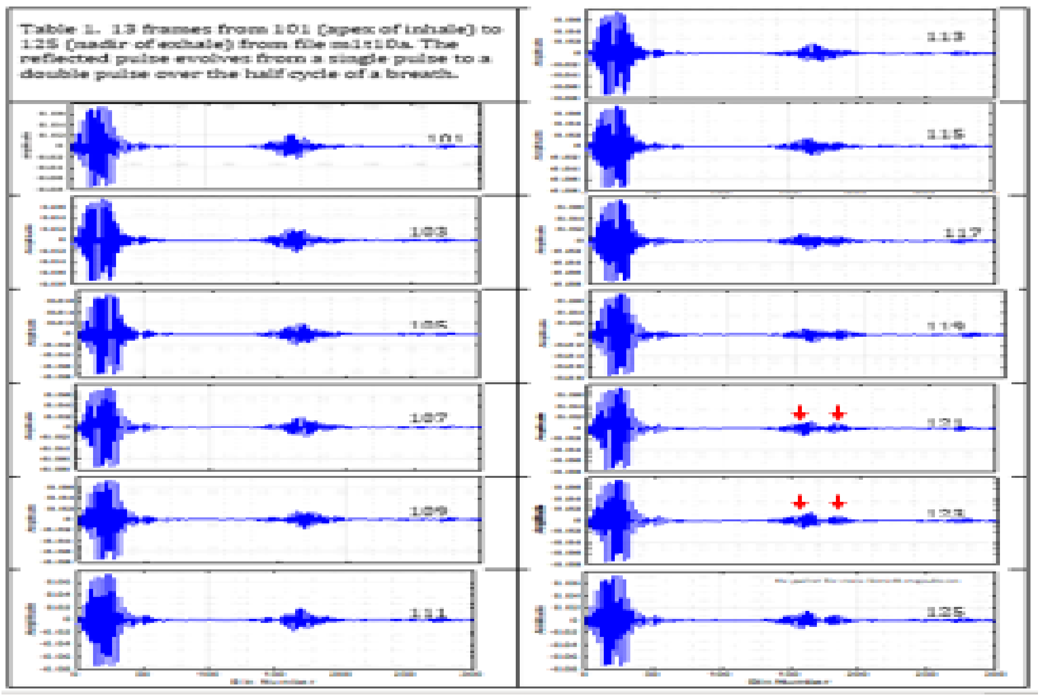


Table 1. 13 frames from 101 (apex of inhale) to 125 (nadir of exhale) from file m1t10a. The reflected pulse evolves from a single pulse to a double pulse over the half cycle of a breath.

Fig. 5-B. Reflected pulse evolution while breathing.

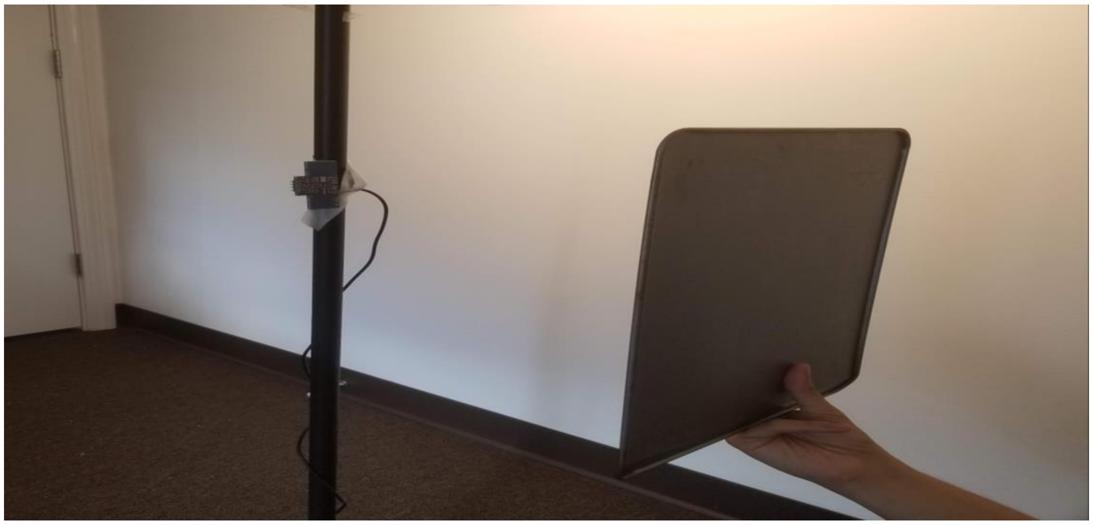


Fig. 5-C. Setup for metal sheet experiment.

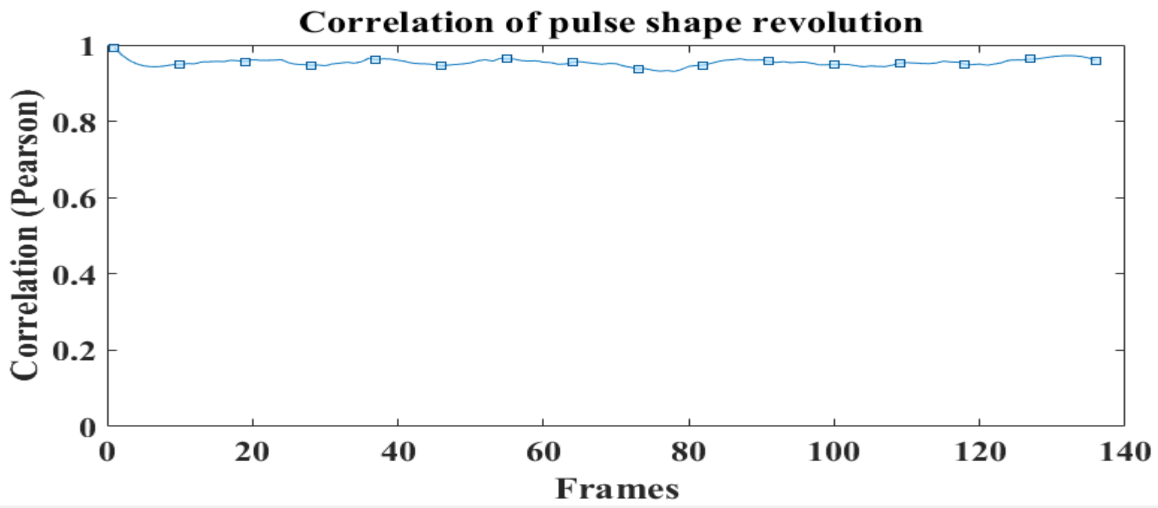


Fig. 5-D. Correlation measures for pulse shape.

The UWB radar reflection pulse recorded from a human body, specifically, from the torso are complex, mainly due to the change of the torso thickness during respiration. In particular, the distance between the chest surface and back surface increases during inhaling as the lungs expand with filling air; while in exhaling, the air goes out and the lungs volume shrinks. As a result, the pulse width corresponding to the time interval between chest surface reflection and back reflection during inhaling and exhaling varies, leading to pulse shape variations. Whereas for the aluminum sheet experiment, as the aluminum sheet maintains its shape consistently, the resulting reflection pulse preserves its pulse width or pulse shape, except the pulse amplitude variation in proportion to the distance between the sheet's surface and the radar. For the reflection signal from the torso, the apparent width of the reflected pulse is generally larger than that of the transmitted pulse, due to the multiple reflections from the human body. In addition, the apparent width appears to change during the breathing cycle. The shape can vary from a single peak, relatively strong pulse to a pulse having multiple unsymmetrical peaks.

When trying to analyze the received signal from the human body, we created the pulse shapes seen by the radar when reflecting from the human subject during respiratory. First, we built a replica of the transmitted pulse that is a sine wave tapered by a Gaussian which is known as Morlet wavelet. In Fig 5-E, we can see the transmitted signal collected from the digital chip of the radar and the one constructed. The Tx pulse is the product of a sine wave modified by a Gaussian envelope. Analytically, this has the following form:

$$f_{tx}(x) = a * \ e^{-.5\left(\frac{x-b}{c}\right)^2} \sin(d * x) \qquad (5)$$

with the four parameters *a* = magnitude, *b* = Gaussian mean, *c* = Gaussian standard deviation, and *d* = sinusoidal frequency.

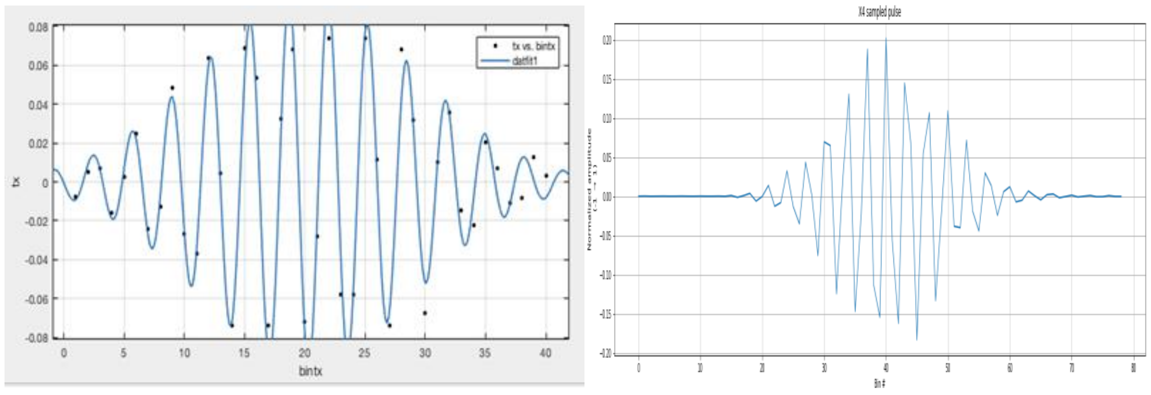


Fig 5-E. Constructing the transmitted pulse

After that, we built a replica of the received pulse based on the transmitted pulse characteristics. The Tx pulse is fit first to verify the frequency and the standard deviation. The model for the Rx pulse is a sum of two or more pulses of the type shown in Eq.5. However, there are two types of impacts the sum pulses generates. The first type is called constructive summation (overlap) where two similar pulses (blue and red) with a nominal separation and almost in phase combine in positive additivity as shown in Fig.5-F. The second type is called destructive summation (overlap) here two similar pulses where one pulse is shifted by 0.5 cycles (out of phase) with the blue pulse as shown in Fig.6-G.

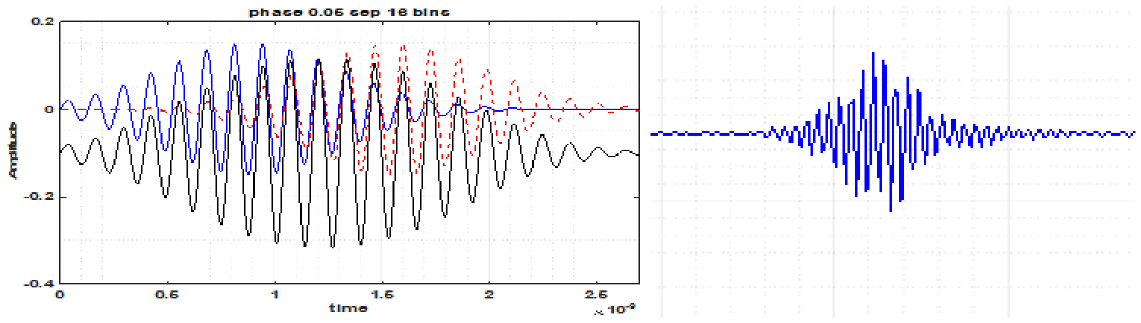


Fig 5-F. Constructive overlap

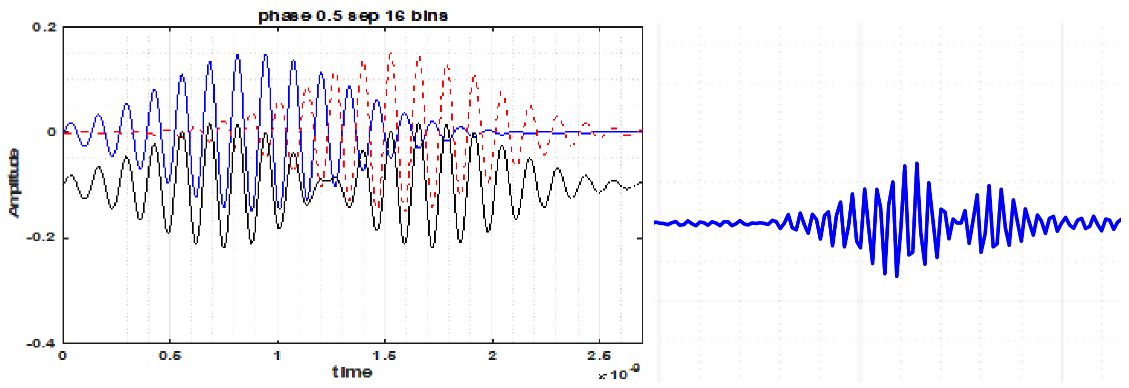


Fig 5-G. Destructive overlap

The received pulse keeps on changing due to the different number of reflections inside the Human body. Since there are multiple ways for the reflected pulse to overlap, it makes difficult to understand and anticipate the shape of the reflected pulse. Therefore, we decided to utilize machine learning to find a fit space for our data and find existing relationships between reflected pulses.

CHAPTER 3: MACHINE LEARNING FOR RESPIRATION DATA PROCESSING

3.1. Machine Learning Model

For this research, we decided to go with different machine learning approaches such as SVM, Logistic Regression, and Random Forest. From there we decided to go with the best one. The Random forest had the best accuracy performance amongst all our models. Below is discussed how every model was constructed.

3.1.1. Logistic Regression

Logistic regression is basically a supervised classification algorithm. In a classification problem, the target variable (or output), y, can take only discrete values for a given set of features (or inputs), X. The algorithm builds a regression model to predict the probability that a given data entry belongs to the category numbered as “1”. Logistic regression models the data using the sigmoid function.

$$g\,(z) = \frac{1}{1+e^{-z}} \qquad (5)$$

The function maps any real value into another value between 0 and 1. In machine learning, we use sigmoid to map predictions to probabilities. The cost function represents an optimization objective. We create a cost function and minimize it so that we can develop an accurate model with minimum error.

$$J(\theta) = -\frac{1}{m}\sum_{i=1}^{m}[y^{(i)}log(h_\theta(x^{(i)})) + (1-y^{(i)})log(1-h_\theta(x^{(i)}))] \quad (6)$$

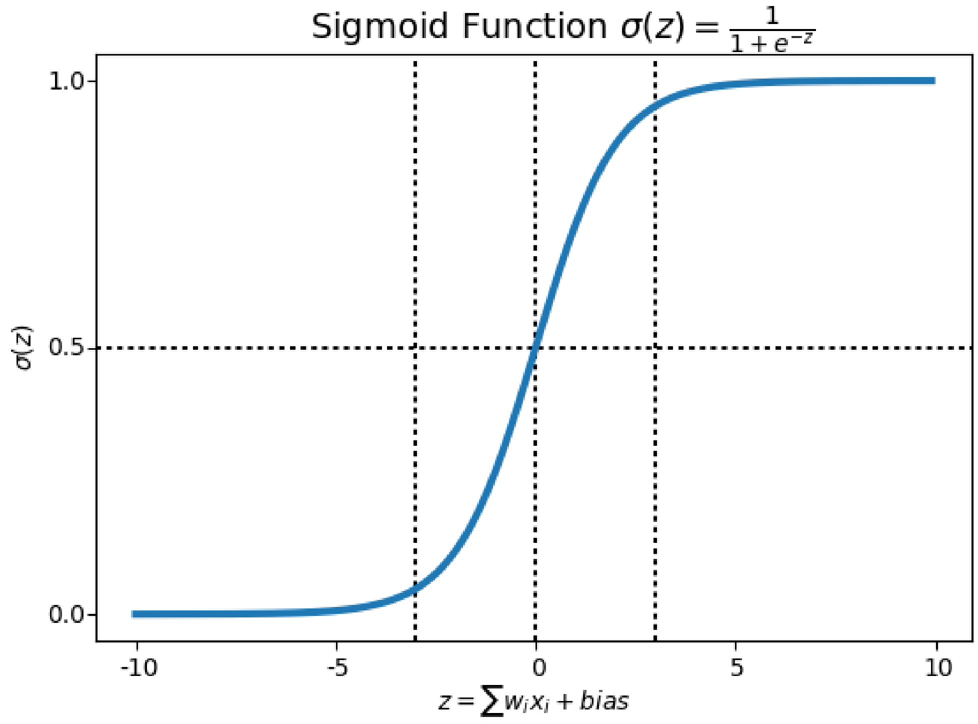


Fig. 6-A. Sigmoid Function.

To reduce our cost function, we are using the Gradient Descent. We are minimizing our cost function by running the gradient descent function on each parameter.

$$\theta_j := \theta_j - \alpha \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) x_j^{(i)} \quad (7)$$

When using logistic regression to classify our pulse shape, our model provided an accuracy of 73%. When the samples that we are trying to classify are highly correlated or highly nonlinear, the coefficients of our logistic regression will not correctly predict the gain/loss from each individual feature.

3.1.2. SVM (Support Vector Machine)

“Support Vector Machine” (SVM) is a supervised machine learning algorithm. In this algorithm, we plot each data item as a point in n-dimensional space (where n is the number of features you have) with the value of each feature being the value of a particular coordinate. Then, we perform classification by finding the hyper-plane that differentiates the two classes very well. The reason we decided to use SVM is its many advantages. The SVM uses a subset of training points in the decision function (called support vectors), so it is also memory efficient. It is also versatile where it can be built using different Kernel functions which can be specified for the decision function.

When designing our SVM, we built it using some of the common kernels at their optimized hyper-parameters.

Kernel	C - Value	gamma	Accuracy
RBF	10	0.01	88%
Linear	1	0.001	83%
Sigmoid	0.1	1	43%
Poly	100	0.001	73%

Table 1. SVM accuracy results.

3.1.3. Random Forest

Random Forest (RF), like its name implies, consists of a large number of individual decision trees that operate as an ensemble. Each individual tree in the random forest spits out a class prediction and the class with the most votes becomes our model's prediction (see Fig. 6-B).

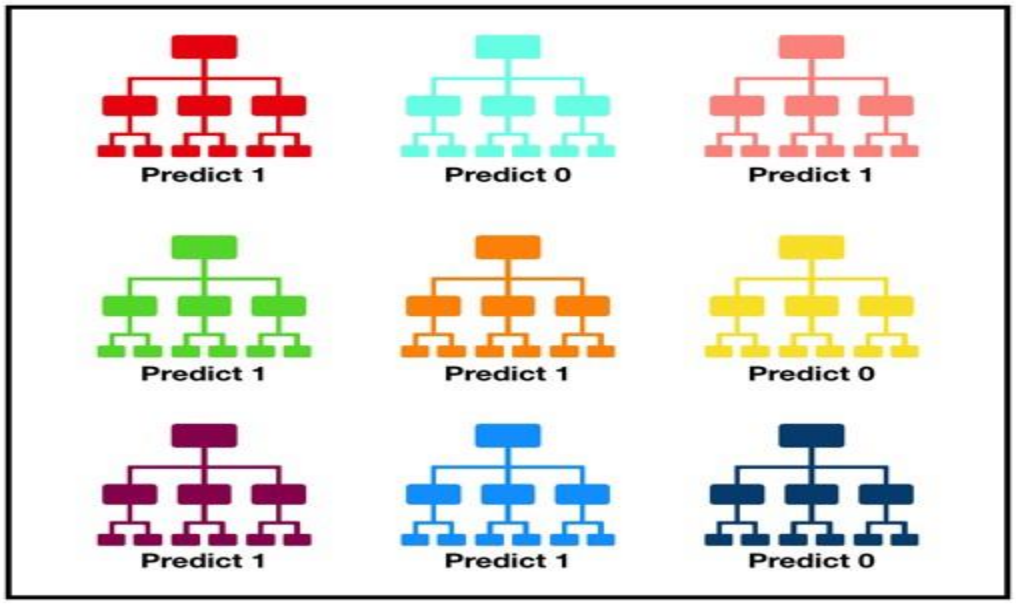


Fig. 6-B. Prediction Tree Sample.

The fundamental concept behind random forest is a simple but powerful one. In data science-speak, the reason that the random forest model works so well is A large number of relatively uncorrelated models (trees) operating as a committee will outperform any of the individual constituent models. The low correlation between models is the key, uncorrelated models can produce ensemble predictions that are more accurate than any of the individual predictions. The reason for this wonderful effect is that the trees protect each other from their individual errors (as long as they don't constantly all err in the same direction). While some trees may be wrong, many other trees will be right, so as a group the trees are able to move in the correct direction.

A random forest is a meta estimator that fits a number of decision tree classifiers on various sub-samples of the dataset and uses averaging to improve the predictive accuracy and control over-fitting. The sub-sample size is always the same as the original input sample size, but the samples are drawn with replacement when bootstrapping. Some parameters are needed to design the estimator:

- **N_estimators:** The number of trees in the forest (n = 100)
- **max_depth:** The maximum depth of the tree (we expanded the nodes until all leaves contain less than min_samples_split samples)
- **min_samples_split:** The minimum number of samples required to split an internal node (minimum = 2)
- **bootstrap:** Whether some samples will be used multiple times in a single tree (we utilized bootstrapping to let some samples will be used multiple times in a single tree.

To evaluate the performance of our machine learning model we need to test it on some unseen data. Based on the model performance on unseen data we can say whether our model is Under-fitting/Over-fitting/Well generalized. Cross-validation (CV) is one of the techniques used to test the effectiveness of machine learning models, it is also a resampling procedure used to evaluate a model if we have limited data. To perform a CV, we need to keep aside a sample/portion of the data on which is not used to train the model, later us this sample for testing/validating. A common technique used for cross-validation is K-fold cross-validation. The K-fold technique results in a less biased model compared to other methods. Because it ensures that every observation from the original dataset has the chance of appearing in training and test set. We start by splitting the entire data randomly into K-folds. Then fit the model using the K — 1 (K minus 1) folds and validate the model using the remaining Kth fold. We repeat this process until every K-fold serves as the test set. Then take the average of your recorded scores. That will be the performance metric for the model.

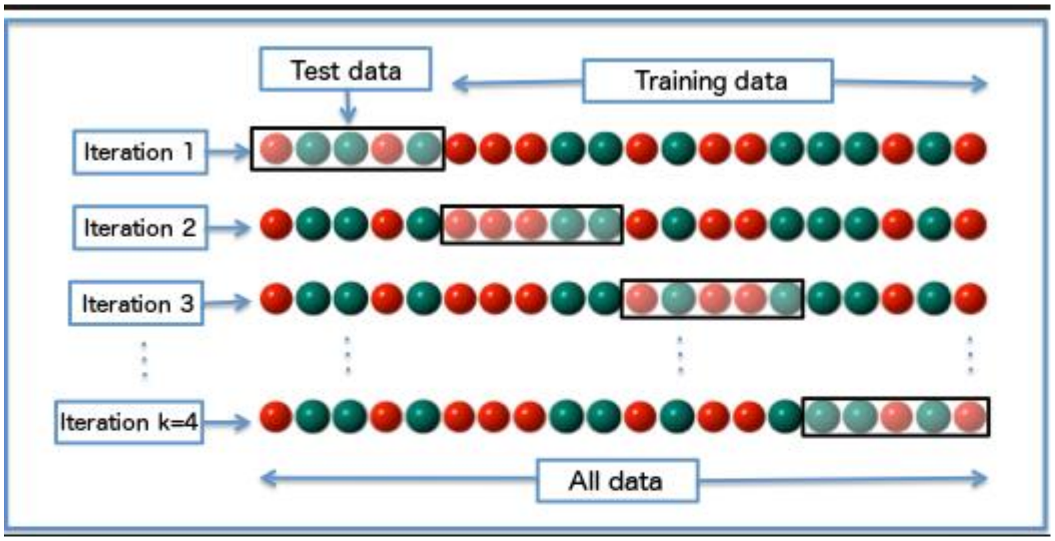


Fig. 6-C. K-fold diagram.

After building our Random Forest to fit the data, we were able to achieve an accuracy of 93%. Furthermore, we were able to test unseen samples of breathing cycles and detect the respiratory rate correctly.

3.2. Experiment

In our experiments, 2200 data frames have been collected from measuring the respiratory of 18 different adults. Every frame represents a pulse shape captured at a speed of 7 frames/s. For each frame, the distance is represented using a unit labeled as "bin" which corresponds to ~1 cm. The respiratory rate for each human subject is measured by radar for 60 seconds while the subject stands in front of radar and is allowed to freely control the breathing. The radar is placed pointing at the subjects' chests at different distances not exceeding 4 meters.

During the 60-second measurement period, data frames are labeled according to the breathing status. Specifically, the frames are labeled as two classes (Empty and Full). As Fig. 6-D shows, at the end of the inhaling stage (maximum volume of lungs), the data frame is labeled as "Full"; while at the end of the exhaling stage (minimum volume of lungs), the corresponding data frame is labeled as "Empty". For the element pulses in each data frame, they are characterized by three featuring parameters which include the pulse peak value, the pulse width, and the distance from the subject to the radar.

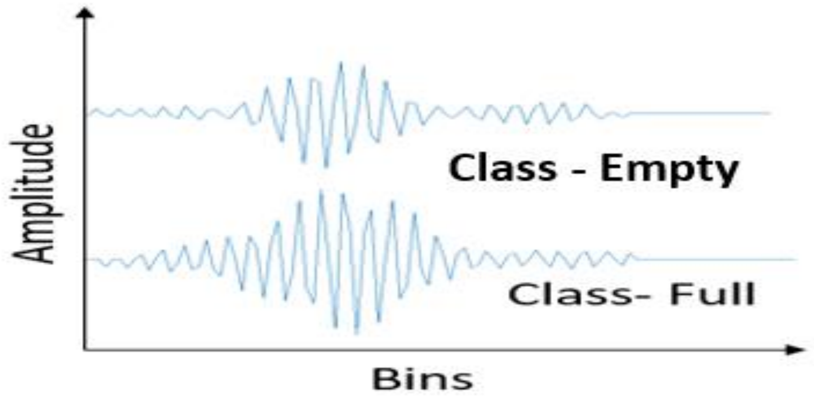


Fig. 6-D. Two classes [Empty & Full] that define breathing.

We used the same dataset for the three ML algorithms. Also, we used cross-validation to test all our models. Since RF provided the best accuracy results out of all the algorithms. We decided to go with it as our algorithm of choice. The RF forest provided consistency and adaptability when shuffling the data. Moreover, we tested the model constructed by the RF with full breathing samples and it was able to detect the breathing rate accurately.

For each decision tree in our RF model, the split across classes is measured at each node split using the Gini impurity index. The Gini impurity index is computed as:

$$G = \sum_{i=1}^{C} p_i * (1 - p_i) \qquad (8)$$

where 'C' is the number of classes in the feature and Pi is the fraction of samples labeled with class *i*. The decision trees are grown to its full length and the ensemble is performed using all decision trees, in the proposed research we found that 100 decision trees achieved the best results.

In addition, RF is capable of providing an importance score for each feature. RF can score the relevance of each feature through the Gini impurity index. We calculated the feature importance by summing the Gini impurity index values for each feature in the dataset over Random Forest trees. These sums are then normalized and ranked to indicate the feature importance index.

After training our random forest model with 100 trees in the forest, we are able to achieve an accuracy of 93%. We are able to analyze our feature importance analyses on the processed data. The RF can score the relevance of each feature through the Gini impurity index, as shown in Fig. 6-E. In the figure, we can see the impact of variables taking into account the interaction with other variables.

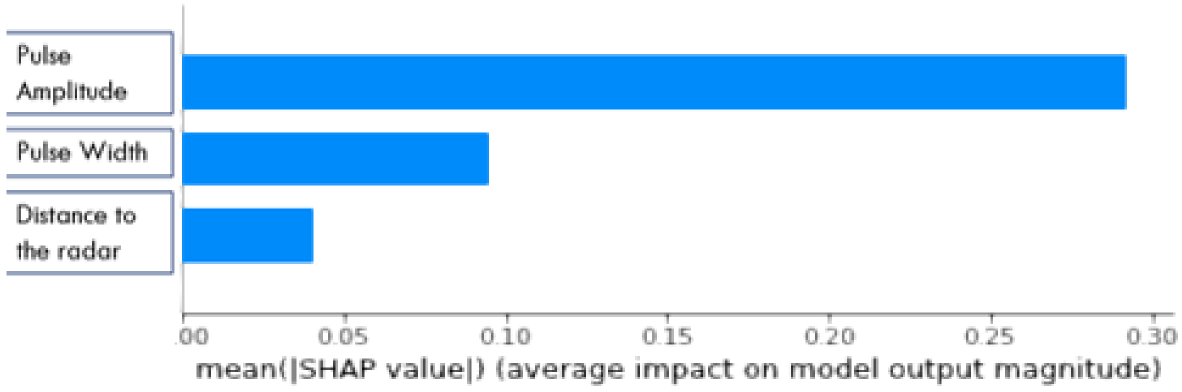


Fig. 6-E. Ranked feature importance (via the Gini method).

Initially another feature such as the subject thickness (torso) was used as one of the features to describe the class of the pulse. However, there was not much variation between the subject used in the study as they were all within similar sizes. The accuracy of the model did not change whether the subject thickness feature was provided or not. Therefore, we did not account for subject thickness in our model.

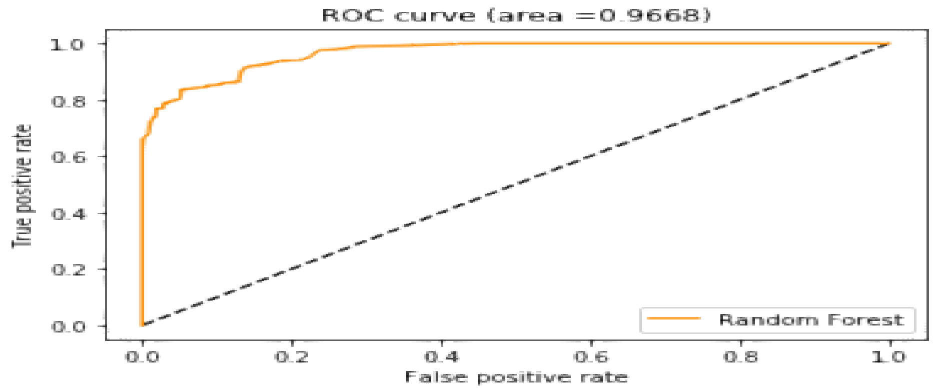


Fig. 6-F. ROC curve.

The receiver operating characteristic (ROC) curve is a performance measurement for classification problems at various threshold settings. ROC is a probability curve, and the area under the curve represents the degree or measure of separability. The ROC tells how much the model is capable of distinguishing between classes.

In Fig. 6-F, each point on the ROC curve represents a sensitivity/specificity pair corresponding to a particular decision threshold. The test has very high discrimination where the ROC curve passes through the upper left corner (96.7% sensitivity). This shows the high accuracy of the model.

CHAPTER 4: EXPERIMENTAL RESULTS

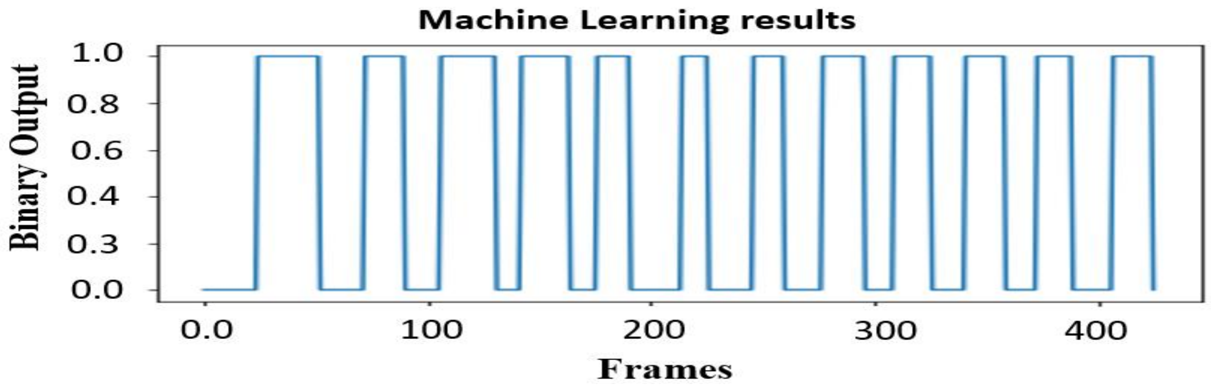
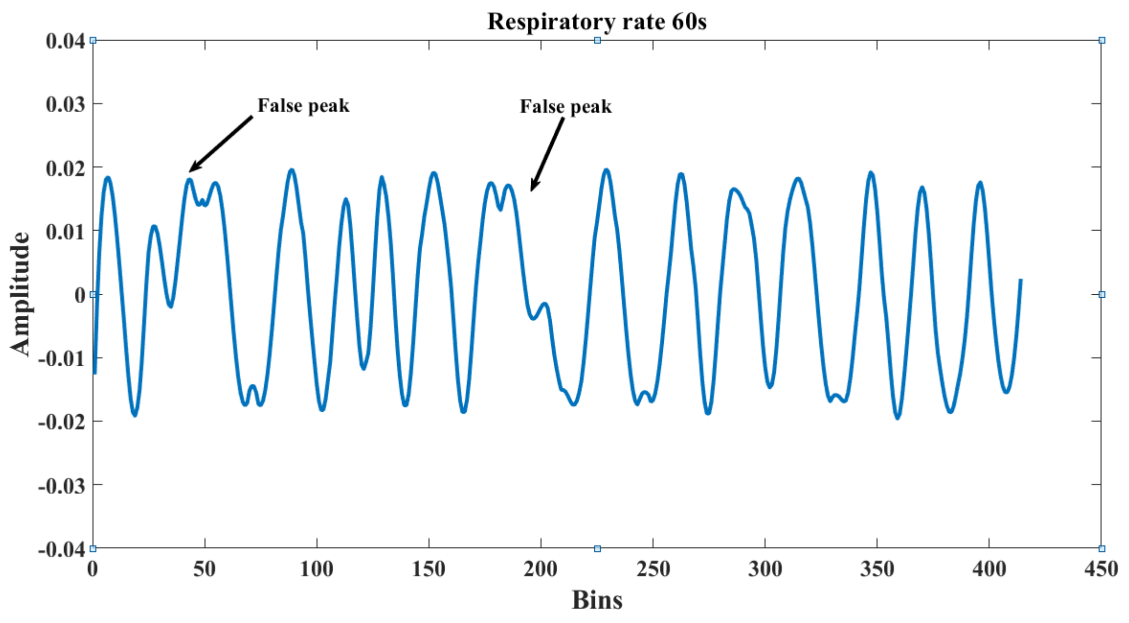


Fig. 7. RR data processed with machine learning.

In the experiment, the ML Random Forest algorithm takes each captured pulse as a data frame. At the output, binary output "0" or "1" is generated to classify the frame as exhaling frame ("Empty class") or inhaling frame ("Full class"). After that, all the output binary points are plotted to construct the square waves. In Fig. 7, illustrates the ML output results for 60s breathing sampling data, which contain 12 data frames.

When applying some of the pre-existing techniques on the same data sample, the results were not as optimal. In Fig. 7-B, we tried to capture the respiratory using the shift method and the auto-correlation method. When measuring the shift of the pulse as an indicator of when the chest expands and contracts, the shift size is not fixed and might be affected by the destructive overlap of the pulses. For the case of auto-correlation, the algorithm was unable to capture the sinusoidal change in the pulse due to the different changes caused by the constructive and the destructive overlap.



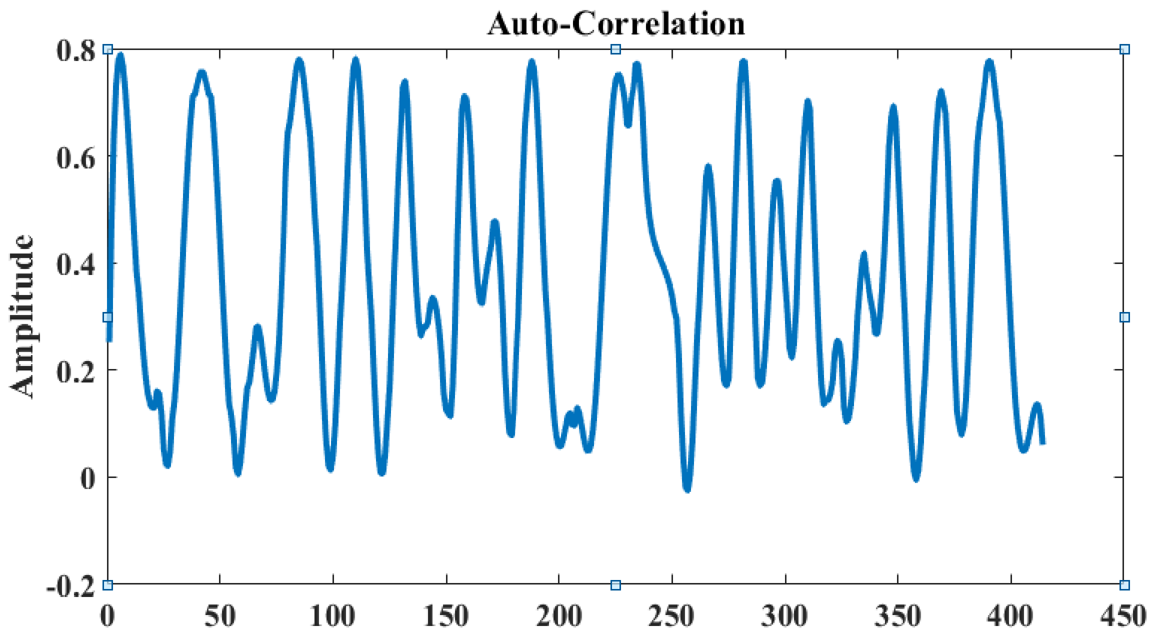


Fig. 7-B. RR data processed with (Pulse shift +Auto-correlation) respectively .

To characterize the respiration rate, frequency analysis is performed. Fig. 8 plots the radar signal spectrum which indicates the detected respiration frequency is 0.2 Hz. This result agrees well with the actual breathing frequency of the test subject who makes 12 breaths during the 60-second test period.

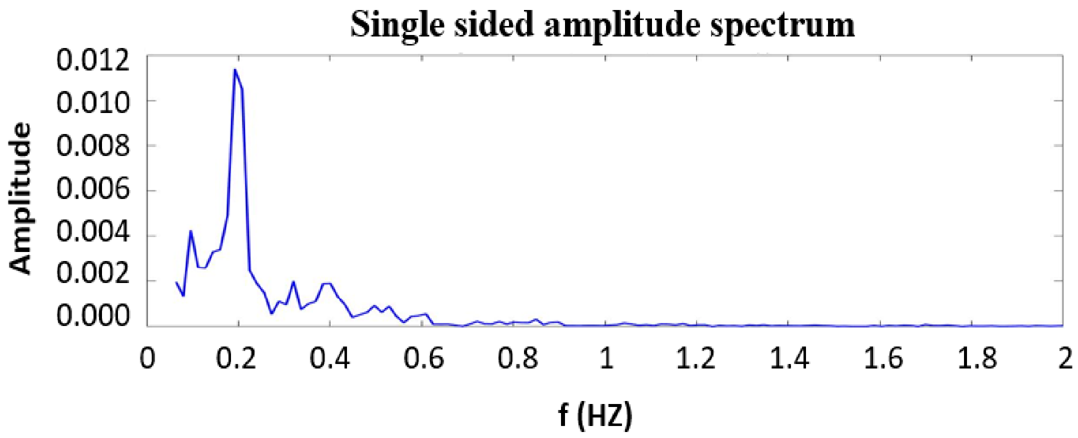


Fig. 8. The frequency response of the analyzed respiratory data.

In our ML approach, the waveform shape evolution of the reflected pulse is explored for respiration rate measurement. For a periodically swinging object of fixed form, its radar reflection pulse maintains a constant shape and the pulse position moves periodically. While for the human subject, as the torso thickness changes in breathing, the radar reflection signal does not maintain a fixed shape. Instead, the pulse width experiences an evolution process, in which the pulse width changes between wide and narrow corresponding to inhalation or exhalation. By examining the pulse shape evolution, the ML algorithm will not mistakenly detect a periodically moving object as the human respiration. For a demonstration, in Fig. 4, an aluminum sheet is used to move periodically back and forth in front of the radar sensor whose moving frequency is close to that of respiration. Adopting the auto-correlation approach in [7], the object's moving pattern is captured and plotted in Fig. 9. However, using this approach, it is not able to identify whether the detected signal is a respiratory signal. While using the ML, as the pulse shape remains unchanged throughout all frames, ML can easily identify that the detected signal is not a respiratory signal. In this test, the reflected pulse is classified as category "1" (Fig. 9) for its shape resembling that of inhaling pulse. Since there is no pulse shape evolution, the signal is detected as a non-respiratory signal.

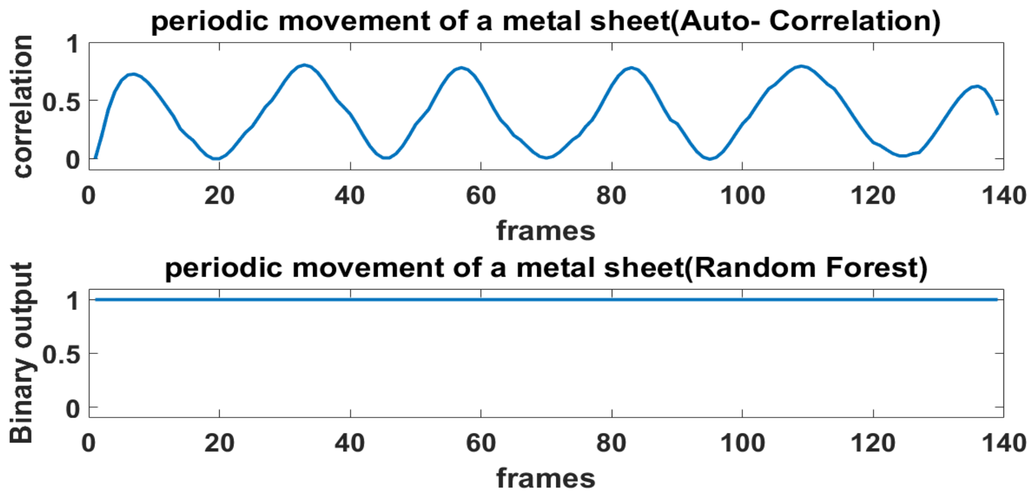


Fig. 9. Object detection.

An additional advantage of our ML approach is that it allows quantizing the pulse width ratio between the inhalation pulse and exhalation pulse. From the health care perspective, the inhalation pulse and exhalation pulse width variation is an implication of the health condition change. For instance, for people of short breaths, as less air is changed in each breath, the respiratory frequency is typically higher than normal, while the pulse width difference between inhalation pulse and exhalation pulse becomes smaller.

For validation, two different patterns of breathing are manipulated in the test, and the corresponding radar waveforms are plotted in Fig. 10-A. In the first waveform, the test subject makes 16 breaths in 60 seconds sampling interval, while in the second waveform, 10 breaths are made in one minute. As shown in Fig. 10-A, the average width ratio between inhalation pulse and exhalation pulse for the faster respiratory is 1.04, while for a slow respirator, it is 1.09. In Fig. 11, the pulse width ratio of all breathing cycles is plotted. We can see that in the deep breathing case that the ratio between the Full pulse and the Bottom pulse is much higher than that of the fast breathing. Therefore, if there is a change in a period of time, it can be detected and examined. The change in pulse width is accompanied by a change in the width of the person. The width of the person is validated using a high precision Meter and an accelerometer by measuring the periodic acceleration.

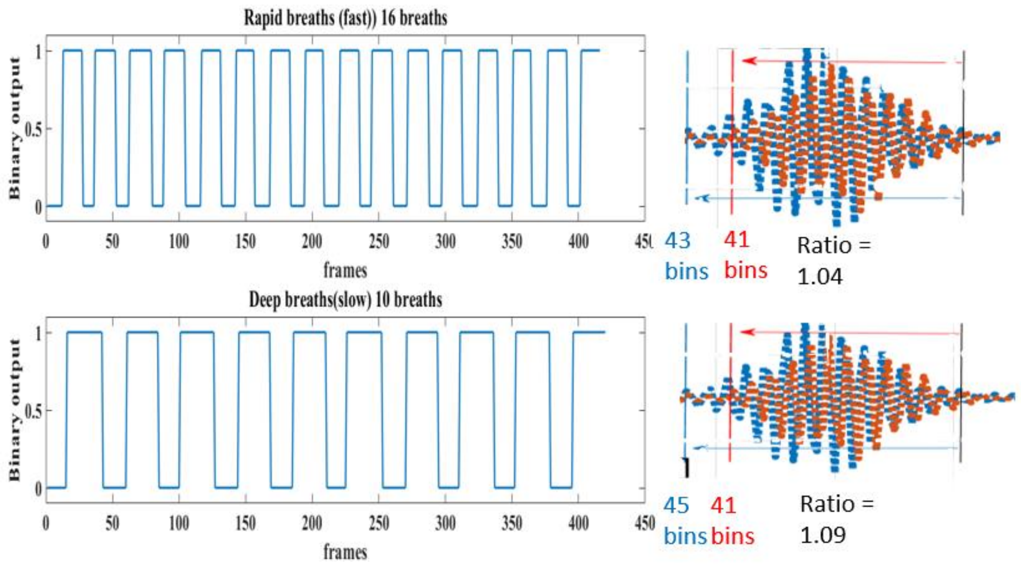


Fig. 10-A. Two breathing samples at different rates.

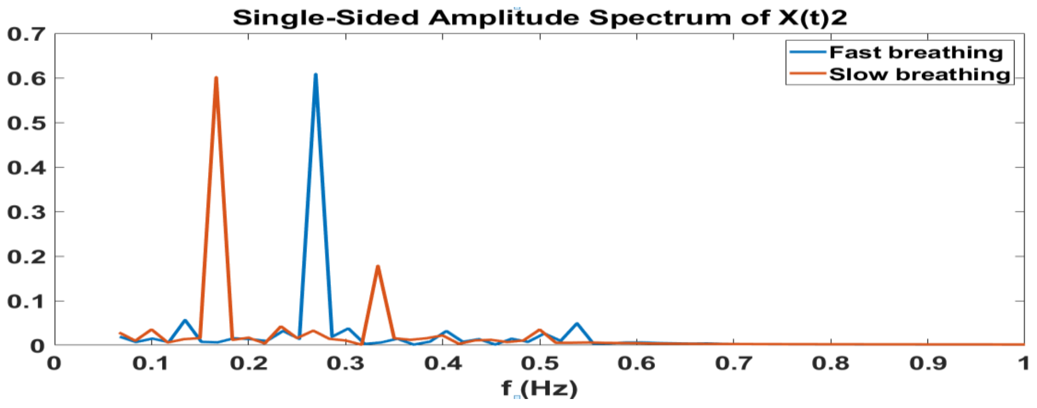


Fig. 10-B. The frequency response of the two breathing samples.

In Fig. 10-B, the spectrum of two respiratory waveforms are plotted. For the faster breathing, the spectrum shows its frequency is 0.17 Hz, while for the slow breathing, the spectrum shows 0.26Hz frequency. These values agree well with actual respiration rates.

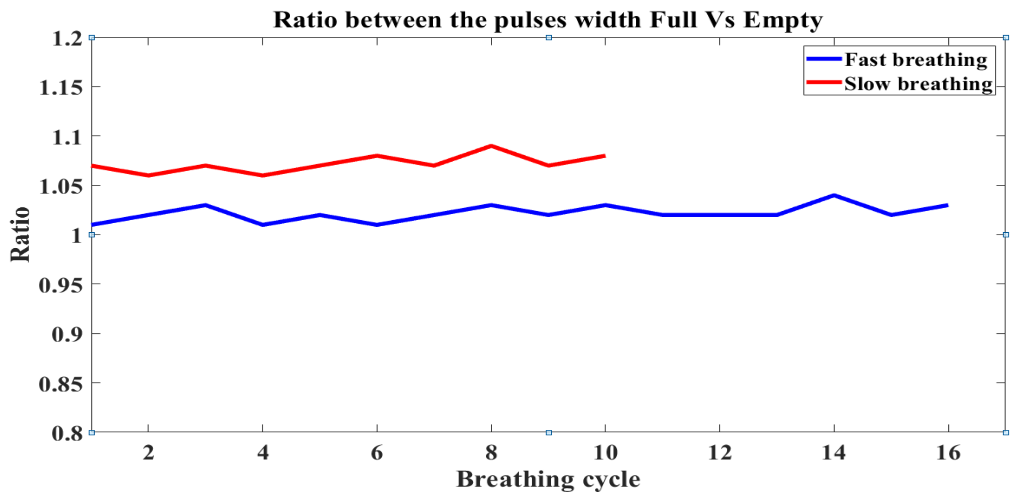


Fig. 11. Pulse width ratio difference between the two samples.

CHAPTER 5: CONCLUSION

In this research, we used a machine learning approach to develop a new technique to utilize a UWB radar for respiration detection and characterization. We are analyzing the reflection of the human body to understand the essential features to understand the pulse. We are describing the pulse by its feature. We are utilizing the random forest classifier to distinguish between pulse’s classes and based on that we are building the respiratory rate signal. Unlike many other approaches presented in the literature, our developed method can track the respiratory waveform evolution so that respiration and other periodically moving objects can be distinguished from each other. In addition, by measuring the ratio of inhalation pulse width and exhalation pulse width, it allows monitoring respiratory pattern variations from deep to rapid breathing, which, in turn, can be utilized to evaluate health condition changes of the human subject under the test.

CHAPTER 6: FUTURE GOALS

- Use Multiple Radar chips with different resolutions to better understand the human body reflection in the radar path. The different variety of resolutions will provide different views of the reflected pulse which might help understand the reflections inside the human body. Furthermore, it will allow us to capture smaller movements like heart vibration. The main challenge is the to find the appropriate hardware with higher level of resolution.

- Build a bigger data set with more features to have a more generalized model. A generalized model could have multiple profiles of data fitting where each subject can have the most optimized profile working on them. New features can be incorporated to raise the accuracy of the model which will open possibilities to function in real time. The challenge with this approach is that its time and money consuming

- Achieve real-time processing using our model. Even though my approach proved successful, real life application will require real-time processing. Real-time processing can be challenging in terms. A new system has to be built to manage the data. The data be acquired, process, and then reported. For all of that to happen smoothly, I expect some C or C++ will be needed to save time. This will require some experience with embedded system programming and data structure.

- Conducting a study to understand reflections from inside the human body. This will allow us to see what truly happens inside the human body in terms of muscle movement. The reflection that occurs inside the human overlap together making them challenging to analyze. Therefore, reconstructing the pulse after their overlapping will allow us to detect where those reflections are occurring. Eventually, we will be able to build a noise map using the radar to see what is inside the human body.

- Conduct a multi-directional body monitoring on the human body. If we can build a map of the inside of the human body. We might be able to see what is inside the human body and avoid using X-rays, MRI's. This research could be very influential. However, this could prove very challenging and very time consuming.

REFERENCES

[1] Wolframm, A.P., and Klausing, H.: “Method for interferometric radar measurement” . *U.S. Patent No. 7,002,508, 21 February 2006*.

[2] X. Huang, L. Sun, T. Tian, Z. Huang, and E. Clancy, “Real-time noncontact infant respiratory monitoring using UWB radar,” in 2015 IEEE 16th International Conference on Communication Technology (ICCT), Hangzhou, China, Oct. 2015, pp. 493–496.

[3] G. R. Wang, H. G. Han, S. Y. Kim, and T. W. Kim, “Wireless vital sign monitoring using penetrating impulses,” *IEEE Microw. Compon. Letters*, vol. 27, no. 1, pp. 94–96, Jan. 2017

[4] L. Liu, Z. Liu, H. Xie, B. Barrowes, and A. C. Bagtzoglou, “Numerical simulation of UWB impulse radar vital sign detection at an earthquake disaster site,” *Ad-Hoc Networks.*, vol. 13, pp. 34–41, Feb. 2014.

[5] W.-P. Hung, C.-H. Chang, and T.-H. Lee, “Real-time and noncontact impulse radio radar system for µm movement accuracy and vital-sign monitoring applications,” *IEEE Sensors Journal*, vol. 17, no. 8, pp. 2349–2358, Apr. 2017.

[6] Andrey B. Borzov, Konstantin P. Likhoedenko, Grigory M. Seregin, Victor B. Suchkov, “Applications of short-range radars on the base of ultrashort pulse single chip,” Moscow, Russia, vol. A247, 2017 IEEE 3rd International Conference on Control Science and Systems Engineering.

[7] Hongming Shen, Chen Xu, and Yongjie Yang, “Respiration and Heartbeat Rates Measurement Based on Autocorrelation Using IR-UWB Radar,” *IEEE Transactions on circuits and systems*, vol. 65, no.10, pp. 1470–1478, Oct. 2018.

[8] Fadel Adib, Zachary Kabelac, Hongzi Mao, Dina Katabi, Robert C. Miller,”Real-time Breath Monitoring Using Wireless Signals”, 2014 Proceedings of the 20th annual international conference on Mobile computing and networking.

[9] K. Ota, Y Ota, M. Otsu, and A. Kajiwara, "Elderly -care motion sensor using UWB-IR," in IEEE Sensors Applications Symposium (SAS), pp. 159-162,2011.

[10] A. Lazaro, D. Girbau, R. Villarino, and A. Ramos, "Vital signs monitoring using impulse based uwb signal," in 41st European Microwave Conference (EuMC), pp. 135-138, 2011.

[11] N. Andre, S. Druart, P. Gerard, R. Pampin, L. M oreno-Hagelsieb, T. Kezai, L. Francis, D. Flandre, and J.-P. Raskin, "Miniaturized wireless sensing system for real-time breath activity recording," IEEE Sensors Journal, vol. 10, no. 1, pp. 178-184,2010.

[12] Y Xu, S. Dai, S. Wu, Y Dang, J. Chen, and G. Fang, "Novel detection method and parameters analysis of vital signal for ultra-wideband radar," in IEEE International Geoscience and Remote Sensing Symposium (IGARSS), pp. 1822-1825,2011.

APPENDIX

The code:

- RF

```
#!/usr/bin/env python
# coding: utf-8
# import Libararies
import os
import scipy
import matplotlib
import numpy as np
import random
import scipy.signal
import statistics
import math
import cv2
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
from sklearn.model_selection import train_test_split
from mlxtend.plotting import plot_decision_regions
import shap
import numpy as np
from lime.lime_text import LimeTextExplainer
from sklearn import metrics
from sklearn.metrics import roc_curve, auc
from sklearn.preprocessing import OneHotEncoder
from sklearn.metrics import roc_auc_score, roc_curve, auc, classification_report

#Load data
file_full=pd.read_csv("/Users/anwer/Desktop/copy/Full.csv", index_col = None, header = None)
file_bottom=pd.read_csv("/Users/anwer/Desktop/copy/Empty.csv", index_col = None, header = None)
file_full = file_full.transpose()
file_bottom = file_bottom.transpose()

print(file_bottom, file_full)
#Data split
file_full2 = []
```

```
file_Empty2 = []

for i in range(1110):
file_full2.append([(file_full[0][i])/max(file_full[0][:]),file_full[1][i],file_full[2][i],[1]])
for i in range(1110):
file_Empty2.append([(file_bottom[0][i])/max(file_full[0][:]),file_bottom[1][i],file_bottom[2][i],[0]])


data = file_full2 + file_Empty2
random.shuffle(data)

#devide features
x = [[each[0],each[1],each[2]] for each in data]
y = [[each[3]] for each in data]

X_train,X_test,Y_train,Y_test = train_test_split(x,y,test_size=0.2,random_state=42)
    # splitting data into train and validation

Y_train = np.reshape(Y_train, (len(Y_train), 1))
Y_test = np.reshape(Y_test, (len(Y_test), 1))

#use the classifier
from sklearn.ensemble import RandomForestClassifier

clf = RandomForestClassifier(n_estimators=100, max_depth=5, random_state=0)
clf.fit(X_train, Y_train.ravel())
#plot ROC curve
one_hot_encoder = OneHotEncoder()
rf_fit = clf.fit(X_train, Y_train)
fit = one_hot_encoder.fit(clf.apply(X_train))
y_predicted = clf.predict_proba(X_test)[:, 1]
false_positive, true_positive, _ = roc_curve(Y_test, y_predicted)

plt.figure()
plt.plot([0, 1], [0, 1], 'k--')
plt.plot(false_positive, true_positive, color='darkorange', label='Random Forest')
plt.xlabel('False positive rate')
plt.ylabel('True positive rate')
```

```
auc = roc_auc_score(Y_test, y_predicted)
plt.title('ROC curve (area =0.9668)')
plt.legend(loc='best')
plt.show()


#make shap plot
explainerRF = shap.TreeExplainer(clf)
shap_values_RF_test = np.asarray(explainerRF.shap_values(np.asarray(X_test)))
shap_values_RF_train = np.asarray(explainerRF.shap_values(np.asarray(X_train)))

print(shap_values_RF_test)

# df_shap_RF_test = pd.DataFrame(shap_values_RF_test[0], columns=[1,2,3])
# df_shap_RF_train = pd.DataFrame(shap_values_RF_train[0], columns=[1,2,3])

j=0
shap.initjs()
print(explainerRF.expected_value)
shap.summary_plot(shap_values_RF_test[j], np.array(X_test).astype(int))
shap.summary_plot(shap_values_RF_test[j], np.array(X_test).astype(int), plot_type="bar")

# Here we test the accuracy of our model

correct = 0
for i in range(len(X_test)):
    if (clf.predict([X_test[i]]) == Y_test[i]):
        correct += 1

print (correct/float(len(X_test))) # this will print out the accuracy of the model.
print(x,y)
value = 1.5
width = 0.75
plot_decision_regions(np.asarray(x), np.asarray(y).flatten(), clf=clf, filler_feature_values={2: value}, filler_feature_ranges={2: width}, legend=2)
plt.show()

```

```
109- fpr, tpr, thresholds = metrics.roc_curve(y, scores,
   pos_label=2)

  -  Keras
  import numpy as np
  import pandas as pd
  import sys
  from keras.models import Sequential
  from keras.layers import Dense
  from sklearn.preprocessing import StandardScaler
  from keras.layers.normalization import BatchNormalization
  from keras.layers import Dropout

  file_full=pd.read_csv("/Users/anwer/Desktop/copy/Full_new.csv"
  , index_col = None, header = None)
  file_bottom=pd.read_csv("/Users/anwer/Desktop/copy/Empty_new.c
  sv", index_col = None, header = None)
  file_full = file_full.transpose()
  file_bottom = file_bottom.transpose()

  # we didn't use those.
  train=[]
  train_targets=[]
  test=[]
  test_targets=[]
  p=[]
  q=[]



  # We will generate train data using 50% of full data and 50%
  of bottom data.
  #is train target for labeling ? yes for train data
  # here I am saying take the first len(file_full)//2 rows
  starting from the start
  # and the all the columns ":" means all
  train_df = file_full.iloc[:len(file_full)//2,:]

  labels=[ 0 for i in range(len(file_full)//2)]

  train_df=train_df.append(file_bottom[:len(file_bottom)//2])

  for i in range(len(file_bottom)//2):

      labels.append(1)
```

```
train_df['label']=labels

#train = train_df.drop('label',axis=1)

train_label= train_df['label']


test_df = file_full.iloc[(len(file_full)//2)+1:len(file_full)]

labels2=[ 0 for i in range(len(file_full)//2)]

test_df=test_df.append(file_bottom[(len(file_bottom)//2)+1:len
(file_bottom)])

for i in range(len(file_bottom)//2-1):

    labels2.append(1)

test_df['label']=labels2

#test = test_df.drop('label',axis=1)

test_label= test_df['label']


# Randomizing the data
train_df = train_df.sample(frac = 1)
test_df = test_df.sample(frac =1)


# Training Model

#can you comment those lines? I want to know what they do?
#train = np.array(train).reshape(-1,1)
scaler = StandardScaler()

train=scaler.fit_transform(train_df.iloc[:,:-1])
test=scaler.fit_transform(test_df.iloc[:,:-1])

# Creating Deep Model



model = Sequential()

# Add an input layer
```

```
model.add(Dense(320, activation='relu', input_shape=(3,)))

# Add one hidden layer
model.add(Dense(320, activation='relu'))
model.add(BatchNormalization())

# Add an output layer
model.add(Dense(1, activation='sigmoid'))

#add improvements

model.add(Dropout(0.2))
#Train the model

model.compile(loss='binary_crossentropy',optimizer='adam',metr
ics=['accuracy'])

model.fit(train,train_df.iloc[:,-1],epochs=50, batch_size=50,
verbose=1)

#TEst the model

y_pred = model.predict(test_df.iloc[:,:-1])

# Evaluate the model


score = model.evaluate(test, test_df.iloc[:,-1])

print(score)
###############################################################
##
#tester=pd.read_csv("/Users/anwer/Desktop/copy/tester.csv",
index_col = None, header = None)
#tester = tester.transpose()
#y_pred = model.predict(test_df.iloc[:,:-1])
```

- RF with images

```
# -*- coding: utf-8 -*-
"""
Created on Thu Apr  4 11:47:15 2019

@author: anwer
"""

#!/usr/bin/env python
```

```
# coding: utf-8

# In[1]:


#cell 1
# First we import the necessary libraries
import os
import scipy
import matplotlib
import numpy as np
import random
import scipy.signal
import statistics
import math
import cv2 # you need to install opencv library, coz it will
help in
           # getting the peaks in the image
import matplotlib.pyplot as plt
import numpy as np
from sklearn.model_selection import train_test_split


# In[2]:


# Each image has a mid line from where the peaks start.
# we are trying to detect the row number of that line in the
image.
# The way we are detecting it is through the colour pixels in
the image.
# Each blue colour pixel in the image has the RGB values
(0,114,189)
# so if a row contains more than 50 pixels consecutively which
contains this
# colour, we conclude that it is the middle line.

def get_mid_line(image):
    count = 0
    for y in range(len(image)):
        for x in range(len(image[y])):
            if (np.array_equal(image[y][x],
np.array([0,114,189]))):
                count += 1
            else:
                count = 0
            if (count >= 50):
                return y


# In[3]:


# now we will get the sum of the peaks in the given image
# To do this we use opencv's cornerHarris function.
# this function gives the corners in an image which in our
case is peaks.
# we get the coordinates of these peaks through the
cornerHarris function,
# and we subtract it from the coordinates of mid line in order
to find the
# height of the peak. Then we add all these heights to get the
sum of the
# heights which is our feature for classifying the image.

def give_peak_sum(file):
    image = cv2.imread(file) # opencv's image read function
    image_copy = cv2.cvtColor(image, cv2.COLOR_BGR2RGB) #
converts image from
    # BGR color space to RGB color space

    image_dims = image.shape
    x_dim = image_dims[1]

    # converting to gray scale
    gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    gray = np.float32(gray)

    # detect corners
    dst = cv2.cornerHarris(gray, 2, 3, 0.04)

    # dilate corner image to enhance corner points
    dst = cv2.dilate(dst, None)

    thresh = 0.02*dst.max()

    peak_sum = 0
    mid_line = get_mid_line(image_copy) # using the previously
defined function

    for j in range(0, dst.shape[0]):
        for i in range(0, dst.shape[1]):
            if (dst[j, i] > thresh and i < 0.4*x_dim):
                peak_sum += abs(j-mid_line)

    return (peak_sum)


# In[4]:


# Here we are just collecting the data that we have

full_images = []
bottom_images = []

for file in
os.listdir("C:/Users/anwer/Desktop/copy/machine/full"): # I
don't know the path of the data files
                                # in your system, modify the
path accordingly.
    full_images.append(file)


for file in
os.listdir("C:/Users/anwer/Desktop/copy/machine/bottom"):
    bottom_images.append(file)

print (len(full_images), len(bottom_images))


# In[5]:


# Here for each image, we will get the sum of peaks. This is
the main
# training step, it might take around 30-45 minutes to run
depending on your
# system capabilities

data_full = []
#data_full3=[]
data_bottom = []
#data_bottom2 = []


for file in full_images:
```

```
data_full.append((give_peak_sum("C:/Users/anwer/Desktop/copy/m
achine/full/"+file),0))

#data_full2.append((give_peak_sum("C:/Users/anwer/Desktop/copy
/machine/full/"+file)))
data_full2 = []
for i in range(len(data_full)):
    data_full2.append(data_full[i][0])

print(data_full2)


    #print(file)


for file in bottom_images:

data_bottom.append((give_peak_sum("C:/Users/anwer/Desktop/copy
/machine/bottom/"+file), 1))

#data_bottom2.append((give_peak_sum("C:/Users/anwer/Desktop/co
py/machine/bottom/"+file)))
data_bottom2 = []
for i in range(len(data_bottom)):
    data_bottom2.append(data_bottom[i][0])

print(data_bottom2)



print (len(data_full), len(data_bottom))


# In[6]:


# Here we mix the two lists data_full and data_bottom and
shuffle it for
# randomness

data = data_full + data_bottom
random.shuffle(data)


# In[7]:
```

```
# here we take features in x variable and labels in y variable
because
# skikit learn libraries require it differently

x = [[each[0]] for each in data]
y = [[each[1]] for each in data]
print (len(x), len(y))


# In[8]:


# Here we split our dataset of total 228 images into training
and testing
# datasets

x_train, x_test, y_train, y_test = train_test_split(x, y,
test_size=0.2, random_state=42)
print (len(x_train), len(x_test))
print (len(y_train), len(y_test))


# In[9]:


# Here we train our data on a Random Forest Classifier
algorithm

from sklearn.ensemble import RandomForestClassifier

clf = RandomForestClassifier(n_estimators=100, max_depth=5,
random_state=0)
clf.fit(x_train, y_train)


# In[10]:


# Here we test the accuracy of our model

correct = 0
for i in range(len(x_test)):
    if (clf.predict([x_test[i]]) == y_test[i]):
        correct += 1
```

```
print (correct/float(len(x_test))) # this will print out the
accuracy of the model.




# In[ ]:


#here we take the new data set and test image by image and
setermine if it FULL or Bottom
test_set = []
test_result=[]
test_dir= "C:/Users/anwer/Desktop/copy/object/"

for file in sorted(os.listdir(test_dir), key=lambda x:
int(x.split('.')[0][4:])):
    test_set.append((give_peak_sum(test_dir+file), file))
    #print(give_peak_sum(test_dir+file), file)
   # print(give_peak_sum(test_dir+file))
    test_result.append((give_peak_sum(test_dir+file)))
    print(file)



#test_set[0] = [data][file]
classvals = []
for i in range(len(test_set)):
    classval = clf.predict([[test_set[i][0]]])
    classvals.append(classval)
    print(classvals)


# In[16]:


total =[]
for i in range(len(classvals)):
    total.append(classvals[i][0])
print(total)


# In[19]:
```

```
print(test_set[1][0])


# In[ ]:
print(test_result)
print(max(test_result))

print(statistics.mean(test_result))

print(min(data_full2))
print(min(data_bottom2))
print(((statistics.mean(data_full2)+statistics.mean(data_botto
m2))/2)/statistics.mean(test_result))
####################################################
#run the test on new data
new_result= []
new_result = np.array(test_result)
a = new_result*3
print(a)
classvals = []
for i in range(len(a)):
    classval = clf.predict([[a[i]]])
    classvals.append(classval)
print(classvals)
total =[]
for i in range(len(classvals)):
    total.append(classvals[i][0])
print(total)
plt.plot(total)
plt.ylabel('amp')
plt.xlabel('Frames')
plt.show()
# code to flip 0s to 1 and 1s to 0
for i in range(len(total)):
    if total[i] == 0:
        total[i] = 1
    else:
        total[i] = 0

# print total
print(total)
####################################################
#equalizer
lst = total
for i in range(3, len(lst) - 3):
    if lst[i - 1] == 1 and lst[i - 2] == 1 and lst[i - 3] == 1
and lst[i + 1] == 1 and lst[i + 2] == 1 \
            and lst[i + 3] == 1:
        lst[i] = 1
    if lst[i - 1] == 0 and lst[i - 2] == 0 and lst[i - 3] == 0
and lst[i + 1] == 0 and lst[i + 2] == 0 \
            and lst[i + 3] == 0:
        lst[i] = 0
#equalizer
i = 0
while(i<len(lst)):
    count = 0
    j = i+1
    while(j<len(lst)):
        if(lst[i]!=lst[j]):
            break
        count += 1
        j += 1
    if(count <= 5):
        k = i
        while(k<j):
            if(lst[k]==0):
                lst[k] = 1
            else:
                lst[k] = 0
            k += 1
    i = j
print(lst)
###################################################
#filtering
def sgolay2d ( z, window_size, order, derivative=None):
    """
    """
    # number of terms in the polynomial expression
    n_terms = ( order + 1 ) * ( order + 2)  / 2.0

    if  window_size % 2 == 0:
        raise ValueError('window_size must be odd')

    if window_size**2 < n_terms:
        raise ValueError('order is too high for the window
size')

    half_size = window_size // 2

    # exponents of the polynomial.
    # p(x,y) = a0 + a1*x + a2*y + a3*x^2 + a4*y^2 + a5*x*y +
...
    # this line gives a list of two item tuple. Each tuple
contains
    # the exponents of the k-th term. First element of tuple
is for x
    # second element for y.
    # Ex. exps = [(0,0), (1,0), (0,1), (2,0), (1,1), (0,2),
...]
    exps = [ (k-n, n) for k in range(order+1) for n in
range(k+1) ]

    # coordinates of points
    ind = np.arange(-half_size, half_size+1, dtype=np.float64)
    dx = np.repeat( ind, window_size )
    dy = np.tile( ind, [window_size,
1]).reshape(window_size**2, )

    # build matrix of system of equation
    A = np.empty( (window_size**2, len(exps)) )
    for i, exp in enumerate( exps ):
        A[:,i] = (dx**exp[0]) * (dy**exp[1])

    # pad input array with appropriate values at the four
borders
    new_shape = z.shape[0] + 2*half_size, z.shape[1] +
2*half_size
    Z = np.zeros( (new_shape) )
    # top band
    band = z[0, :]
    Z[:half_size, half_size:-half_size] =  band -  np.abs(
np.flipud( z[1:half_size+1, :] ) - band )
    # bottom band
    band = z[-1, :]
    Z[-half_size:, half_size:-half_size] = band  + np.abs(
np.flipud( z[-half_size-1:-1, :] )  -band )
    # left band
    band = np.tile( z[:,0].reshape(-1,1), [1,half_size])
    Z[half_size:-half_size, :half_size] = band - np.abs(
np.fliplr( z[:, 1:half_size+1] ) - band )
    # right band
    band = np.tile( z[:,-1].reshape(-1,1), [1,half_size] )
    Z[half_size:-half_size, -half_size:] =  band + np.abs(
np.fliplr( z[:, -half_size-1:-1] ) - band )
    # central band
    Z[half_size:-half_size, half_size:-half_size] = z
    # top left corner
    band = z[0,0]
    Z[:half_size,:half_size] = band - np.abs(
np.flipud(np.fliplr(z[1:half_size+1,1:half_size+1]) ) - band )
    # bottom right corner
    band = z[-1,-1]
    Z[-half_size:,-half_size:] = band + np.abs(
np.flipud(np.fliplr(z[-half_size-1:-1,-half_size-1:-1]) ) -
band )

    # top right corner
    band = Z[half_size,-half_size:]
    Z[:half_size,-half_size:] = band - np.abs(
np.flipud(Z[half_size+1:2*half_size+1,-half_size:]) - band )
    # bottom left corner
    band = Z[-half_size:,half_size].reshape(-1,1)
    Z[-half_size:,:half_size] = band - np.abs( np.fliplr(Z[-
half_size:, half_size+1:2*half_size+1]) - band )

    # solve system and convolve
    if derivative == None:
        m = np.linalg.pinv(A)[0].reshape((window_size, -1))
        return scipy.signal.fftconvolve(Z, m, mode='valid')
    elif derivative == 'col':
        c = np.linalg.pinv(A)[1].reshape((window_size, -1))
        return scipy.signal.fftconvolve(Z, -c, mode='valid')
    elif derivative == 'row':
        r = np.linalg.pinv(A)[2].reshape((window_size, -1))
        return scipy.signal.fftconvolve(Z, -r, mode='valid')
    elif derivative == 'both':
        c = np.linalg.pinv(A)[1].reshape((window_size, -1))
        r = np.linalg.pinv(A)[2].reshape((window_size, -1))
        return scipy.signal.fftconvolve(Z, -r, mode='valid'),
scipy.signal.fftconvolve(Z, -c, mode='valid')

totall=scipy.signal.savgol_filter(produce,19,4)
z= np.zeros(603)
plt.plot(lst,'r--',z,'b--')
plt.ylabel('amp')
plt.xlabel('Frames')
plt.title('Machine Learning results')
plt.savefig('machine-image.png')
plt.show()
print(lst)
```

- MATLAB

  Communicate with radar and data acquisition

```
%Anwar Elhadad
%biomedical signal processing

%read file and organize into a matrix where each frame is represented
as an
%array

fid = fopen('1 foot sinusoid.txt','rt');
d = cell2mat( textscan(fid, '%f', 'Delimiter','[]', 'collectOutput',
true) );
fclose(fid);

numRow = sum(double(isnan(d)));

numCol = int32(length(d)/numRow);
data = reshape(d,numCol,numRow);
data = data(2:end,:);
%%

%%
frequency = data (100:150,:);
array = frequency(:);
%demodulation = array * cos(
plot(frequency)
% data (120:150,:)
vector = sum(frequency);
%%many point
Z = data(150,:);
z1 = fft(vector);
L=length(vector);
P2 = abs(z1/L);
P1 = P2(1:L/2+1);
P1(2:end-1) = 2*P1(2:end-1);
xaxis= 0.3:0.3:63.3;
f = 0.45*(0:(L/2))/L;
figure(1)
%plot(f,P1)
title('Single-Sided Amplitude Spectrum of X(t)')
xlabel('f (Hz)')
ylabel('|P1(f)|')
%plot(xaxis,Z)
% filtered = bandpass(frequency,[0.08 0.3],Fs)
% %plot(filtered)
% squared = filtered.^2
% squared2 =sum(squared)
% plot(xaxis,squared2)
%%
frequency = data (100:150,:);
array = frequency(:);
T = 6.66;
fs = 1/T;
fc = 8 *(10^9);
%x = array.*cos(2*pi*fc*T);
```

```
%x = demod(array,fc,fs,'am')
x = amdemod(array,fc,fs);
L = length(array);             % Length of signal
t = (0:L-1)*T;        % Time vector
Y = fft(array);
P2 = abs(Y/L);
P1 = P2(1:L/2+1);
P1(2:end-1) = 2*P1(2:end-1);
f = fs*(0:(L/2))/L;
subplot(2,2,1)
plot(f,P1)
title('Single-Sided Amplitude Spectrum of X(t)')
xlabel('f (Hz)')
ylabel('|P1(f)|')
%y = lowpass(x,5,fs)
%plot(x)
L2 = length(x);             % Length of signal
t2 = (0:L-1)*T;        % Time vector
Y2 = fft(x);
P22 = abs(Y2/L2);
P12 = P22(1:L2/2+1);
P12(2:end-1) = 2*P12(2:end-1);
f2 = fs*(0:(L2/2))/L2;
subplot(2,2,2)
 plot(f2,P12)
title('Single-Sided Amplitude Spectrum of X(t)2')
xlim([0 5])
xlabel('f (Hz)')
 ylabel('|P1(f)|')
%plot (y)
%% algorithm
%read the frame
frequency = data (135:185,:);
array = frequency(:);
%plot(array)
for i= 1:139
    %find peaks
  pks = findpeaks((frequency(:,i)));
  %bottom = findpeaks(-1*frequency(:,i));
  %add the values inside a gaussian curve
  absolute = abs(frequency(:,i));
  S(:,i) = sum(absolute);
  %i = i+1;
end
plot(S)
xlabel('bins')
ylabel('area under the curve')
%%
%filter
pong = bandpass(S,[0.1 4],6.67);
plot(pong)
%%
smooth = sgolayfilt(pong,3,11);
plot(smooth)

%%
%equalizer
```

```
for i = 1:length(smooth)
    if smooth(i)>0.007
        smooth(i) =0.007;
    end
    if smooth(i)<0.001
        smooth(i) = 0.001;
    end
end
smooth_more = sgolayfilt(smooth,4,25);
smooth_more= smooth_more*100;
%%
subplot(2,1,1)
plot(smooth_more)
ylim([-0.1 1])
title('periodic movement of a metal sheet(Auto- Correlation)')
ylabel('correlation')
xlabel('frames')
subplot(2,1,2)
XX = ones(1,139);
plot(XX)
ylim([-0.1 1.1])
title('periodic movement of a metal sheet(Random Forest)')
ylabel('Binary output')
xlabel('frames')
%%
plot(smooth_more)
yyaxis left
ylim([-0.04 0.04])
ylabel('Amplitude')
hold on
yyaxis right
plot(smooth_more2)
ylim([-2 2])
title ('Pulse Analysis vs pulse movement [Object detection]')
xlabel('Frames')
ylabel('Amplitude [0 or 1]')

%%
%frequency domain
L = length(smooth_more);             % Length of signal
Y = fft(smooth_more);
P = abs(Y/L);
P1 = P(1:L/2+1);
P1(2:end-1) = 2*P1(2:end-1);
f = 6.67*(0:(L/2))/L;
plot(f(:,5:end),P1(:,5:end))
title('Single-Sided Amplitude Spectrum of X(t)2')
xlim([0 2])
xlabel('f (Hz)')
%%
fs = 6.6;
L = length(S);             % Length of signal
Y = fft(S);
P = abs(Y/L);
P1 = P(1:L/2+1);
P1(2:end-1) = 2*P1(2:end-1);
f = fs*(0:(L/2))/L;
```

```
 plot(f(:,5:end),P1(:,5:end),'*')
title('Single-Sided Amplitude Spectrum of X(t)2')
xlim([0 10])
xlabel('f (Hz)')
ylabel('|P1(f)|')
[M,I] = max(P1);
%%

sample1 = [0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1
            1 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
            0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
            1 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
            0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1
            1 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1
            0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1
            1 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1
            0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1
            1 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
            0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
            1 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
            0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1
            1 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1
            0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1
            1 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1];
sample1=sample1.';
sample1=sample1(:);


sample2 = [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1
            0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1
            0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1  1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1
            0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1
            0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1
            0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1  1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1
            0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1
            0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1  1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1
            0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1
            0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1];
sample2=sample2.';
sample2=sample2(:);

subplot(2,1,1)
plot(sample1)
ylim([-0.1 1.1])
title('Rapid breaths (fast))')
ylabel('Binary output')
```

```
xlabel('frames')
subplot(2,1,2)
plot(sample2)
ylim([-0.1 1.1])
title('Deep breaths(slow)')
ylabel('Binary output')
xlabel('frames')
%%

sample1 = lowpass(sample1,0.1);
L = length(sample1);             % Length of signal
Y = fft(sample1);
P = abs(Y/L);
P1 = P(1:L/2+1);
P1 = (P(1:L/2+1)).';
P1(2:end-1) = 2*P1(2:end-1);
f = 7*(0:(L/2))/L;
plot(f(:,5:end),(P1(:,5:end)))
title('Single-Sided Amplitude Spectrum of X(t)2')
xlim([0 1])
xlabel('f (Hz)')
hold on
sample2 = lowpass(sample2,0.08);
L = length(sample2);             % Length of signal
Y = fft(sample2);
P = abs(Y/L);
P1 = P(1:L/2+1);
P1 = (P(1:L/2+1)).';
P1(2:end-1) = 2*P1(2:end-1);
f = 7*(0:(L/2))/L;
plot(f(:,5:end),(P1(:,5:end)))
title('Single-Sided Amplitude Spectrum of X(t)2')
xlim([0 1])
xlabel('f (Hz)')
legend('Fast breathing','Slow breathing')
%%
%pulse width
Ratio1 = [ 1.01 1.02 1.03 1.01 1.02 1.01 1.02 1.03  1.02 1.03 1.02 1.02
1.02 1.04 1.02 1.03];
Ratio2 = [ 1.07 1.06 1.07 1.06 1.07 1.08 1.07 1.09 1.07 1.08];

plot(Ratio1,'b')
hold on
plot(Ratio2,'r')
legend('Fast breathing','Slow breathing')
title('Ratio between the pulses width Full Vs Empty')
ylim([0.8 1.2])
ylabel('Ratio')
xlim([1 17])
xlabel('Breathing cycle')

%%
x = 0:1:400;
y = 0:1:30;
z = 0:0.01:1;
```

```
[x,y,z] = meshgrid(x,y,z);
v = x.*exp(-x.^2-y.^2-z.^2);

xslice = sample2;    % location of y-z planes
yslice = Ratio2;              % location of x-z plane
zslice = f;         % location of x-y planes

slice(x,y,z,v,xslice,yslice,zslice)
xlabel('x')
ylabel('y')
zlabel('z')
```