NON INVASIVE
INFANT SLEEP APNEA DETECTION
IMPLEMENTATION GUIDE

CO321 CO324 CO325 UNIFIED PROJECT

TEAM

E/14/158: GIHAN JAYATILAKA
E/14/339: SUREN SRITHARAN
E/14/379: HARSHANA WELIGAMPOLA
E/14/237: PANKAYARAJ PATHMANATHAN

Deaprtment of Computer Engineering
Unviersity of Peradeniya

2018

Contents

1. Introduction

This document gives a step by step guide to implement and develop the whole system including the hardware, software, network and security components
The source code can be obtained from :

https://github.com/TeambiteCode/Sleep_Apnea_Detection/tree/master/releases/v1

The system has been tested, and the testing methods have been specified in this document. Nevertheless if there are any issues post them under issues in aforementioned *Github* repository.

2. Abstract

Sleep Apnea is a serious disorder caused by the interruption of breathing during sleep. This can cause the people to stop breathing for several times (even hundreds) if not treated properly. It can affect people of any age. But when the babies are affected with the condition they tend to not get up and keep on sleeping which may risk their lives.
This is a non invasive solution based on video processing. The infant is observed by a video camera which is connected to a single board computer (Raspberry pi) which analyzes the video feed to diagnose breathing anomalies. The camera is turned to a proper orientation for the observation using a robotic arm.

3. Implementation

3.1 Device

This device is packaged with a Raspberry Pi 3B module, Raspberry Pi Camera module and 2 servo motors.

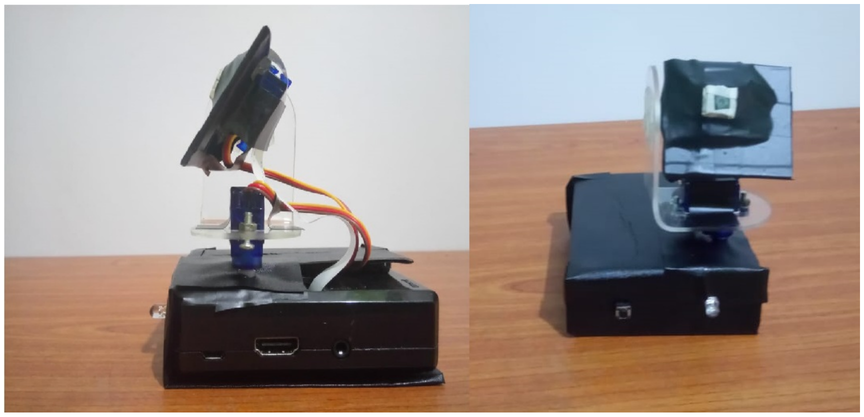

Figure 3.1: Preview of the device

3.1.1 Pre requisites

The requires libraries and programs are installed by the **install.sh**. The following is the list of pre requisites, installation commands and their contribution for advanced users

MQTT

pip install paho-mqtt

Python2.7

If python3 is already installed in the system please remove it before proceeding. This can complicate things when installing python libraries.

Note: Currently opencv3 cannot be installed to python3 in raspberry pi. If it is necessary to install in python3, there is a tedious way to install opencv3 to python3 in raspberry pi 3. This is not recommended!

To uninstall python3,

```
sudo apt-get remove python3
```

Numpy

```
pip install numpy
```

This is used for faster numerical calculations.

Scipy

```
sudo apt-get install python-scipy
```

The pip package manager's release does not have binaries compiled to the raspberry pi architecture. apt has the working version.

Opencv 3

```
sudo apt-get install OpenCV-Python
```

There are numerous ways to install opencv and interface it with python. But most of them lack binaries for the raspberry pi architecture for certain algorithms. Compiling the opencv source by yourself and the apt's release works.

3.1.2 FSM and Run at startup

When the device starts the service, initially it will be in the ready state. In this state the device will take user input through the push down button in order to start the analysis and communicate with the server. When the button is pressed, the device will go to adjust camera state where the module automatically adjust the camera angle to direct it to the infant. When the camera is rotated accordingly, the state will change to analysis. In this state a video of the infant will be recorded and analyzed. The processed information will be sent to the server for further analysis. If the analysis state needs to adjust the camera, the state will be changed and this will be repeated continuously.

3.1.3 Network setting

The network settings are set up when the code is inserted to the node. It is explained in the section **Downloading the software**

3.1.4 Reading input

The device reads the input from the Raspberry pi camera. The camera is directly connected to the device using the camera port in the raspberry pi. To take the user input, a push down button is available to reset the device and start the service. The input from the push down button is connected to the device using GPIO pin 11. In order to remove the unnecessary bouncing effects, 300ms software de-bouncing has been applied.

3.1.5 Rotating Camera

Given different lighting condition the detection of baby by the camera becomes the most primitive task. There are two servo motors connected to the device to turn the camera and adjust the angle of the camera to direct it to the infant. These motors are powered by the raspberry pi itself. Motor shield was not used to power the motors because the angle that the motors rotate are controlled by the software itself with a safe margin. Also the motors does not absorb much power because the camera and the equipment holding the camera is lightweight and the current drown by the power bus in the raspberry pi is capable of supplying the current required for the motors.
The control signal to the servo motors are sent using GPIO pins 12 and 13 in the raspberry pi 3 module. Hardware PWM is used to give the control signal through these pins because using software PWM is consuming CPU and it might not be accurate when other processes are running concurrently. Since the hardware PWM should be controlled by changing the registry values in the raspberry pi, using pure python it is difficult to adjust these parameters. therefore, a third party software *pigpio-daemon* was used. This application should be running in background in order to change the hardware PWM values from python. This can be accomplished by connecting to a port that is used to communicate with the application and sending required commands through that port.

3.1.6 Pin Mapping

Apart from the pins mentioned above, GPIO pin 21 is used to connect the LED as a output to the user to identify the state in which the device is in.

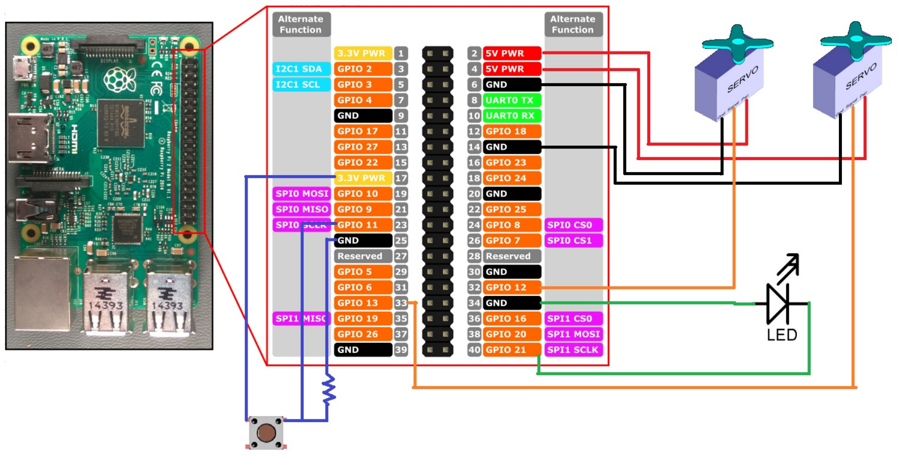

Figure 3.2: Pin mapping of the Raspberry pi

3.1.7 Detecting the breathing pattern

Operation

This part of the program,

- detects the breathing pattern
- analyzes it
- writes it to a detailed pdf report on the device
- transmits the results to the server

Code

The code for this algorithm is present at **https://github.com/TeambiteCode/Sleep_Apnea_Detection/tree/master/Algos**. Following is a detailed description for someone who wants to implement the algorithm themselves in a different language etc:.

Detailed algorithm

The algorithm we propose have several steps.

- The video is taken in as an array of 8 bit unsigned integers.

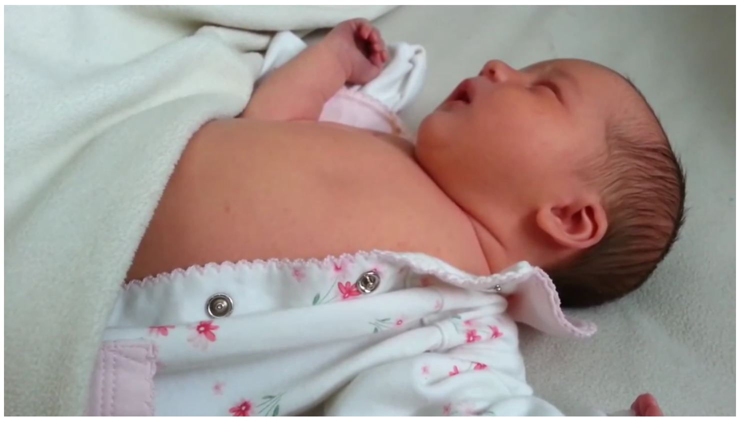

Figure 3.3: The original video

- The video is fed to the **Canny edge detection algorithm**
  The opencv implementation of this algorithm is used. The result is a black and white video stream on where the white corresponds to the edges and black to the rest.

$E_{(t)H \times W}$ is the edge matrix.

$$E_{(t,x,y)} = \left\{ \begin{array}{l} 1 \text{ if } E_{(t,x,y)} \text{ is an edge.} \\ 0 \text{ if } E_{(t,x,y)} \text{ is not an edge.} \end{array} \right\}$$

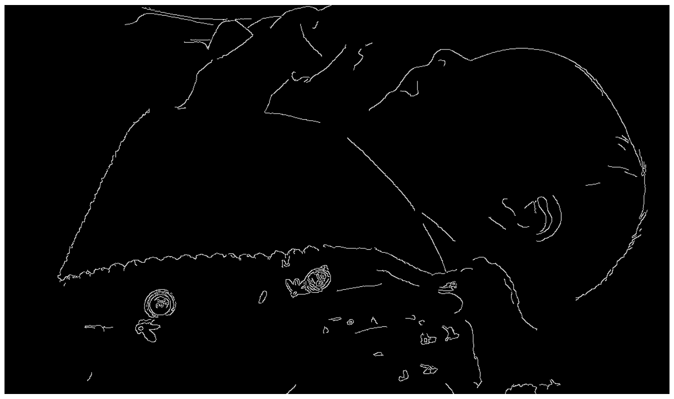

Figure 3.4: The edges of the video

- The region of interest A_0 is chosen.
  $(x, y) \in A_0$
  $x \in [x_0, x_1], y \in [y_0, y_1]$
  As of now our algorithm requires a manual input for this region.

We hope to automate this parameter during the time from 3rd phase evaluation to the final evaluation.

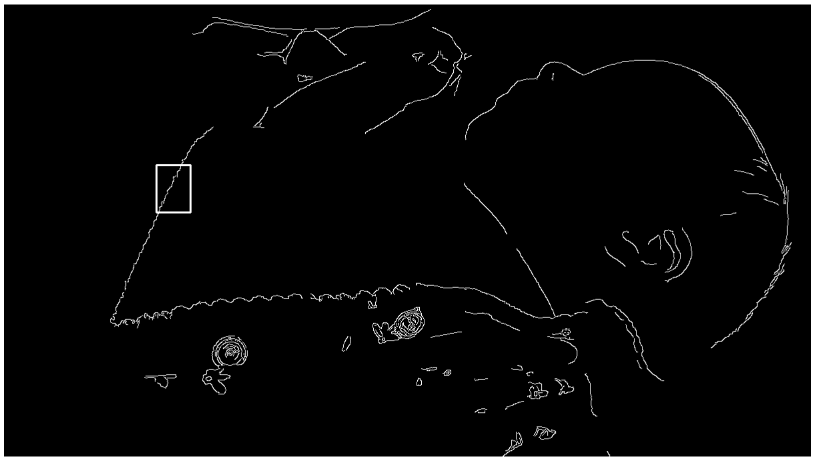

Figure 3.5: The region of interest

- The centroid $C_0(t)$ of the edges in A_0 is calculated for every t
  $C_0(t) = (x_{c_0(t)}, y_{c_0(t)}))$

$$x_{c_0(t)} = \frac{\sum_{(x,y)\in A} E(t,x,y) \times x}{\sum_{(x,y)\in A} E(t,x,y)}$$

$$y_{c_0(t)} = \frac{\sum_{(x,y)\in A} E(t,x,y) \times y}{\sum_{(x,y)\in A} E(t,x,y)}$$

  Special case:
  $(x_{c_0(t)}, y_{c_0(t)}) = (\frac{x_0+x_1}{2}, \frac{y_0+y_1}{2})$ when

$$\sum_{(x,y)\in A} E(t,x,y) = 0$$

- Then the velocity of the centroid $\underline{v}_{(t)}$ is calculated by,
  $\underline{v}_{(t)} = (x_{c_0(t)} - x_{c_0(t-1)})\underline{i} + (y_{c_0(t)} - y_{c_0(t-1)})\underline{j}$
- The direction along which the velocity of the centroid $\underline{v}_{(t)}$ lie is calculated using the **Principle component analysis** as follows,
  Write $v_{(t)}$ as a row vector

$$v_{(t)} = \begin{pmatrix} \underline{\mathrm{v}}_{(t)}.\underline{i} & \underline{\mathrm{v}}_{(t)}.\underline{j} \end{pmatrix}$$

$$v_{(t)} = \begin{pmatrix} \mathrm{v}_{x(t)} & \mathrm{v}_{y(t)} \end{pmatrix}$$

Make a matrix by taking 10 such readings and arranging them as rows,

$$V_{(t)} = \begin{pmatrix} \mathrm{v}_{x(t)} & \mathrm{v}_{y(t)} \\ \mathrm{v}_{x(t-1)} & \mathrm{v}_{y(t-1)} \\ \mathrm{v}_{x(t-2)} & \mathrm{v}_{y(t-2)} \\ .... & ... \\ ... & ..... \\ \mathrm{v}_{x(t-9)} & \mathrm{v}_{y(t-9)} \end{pmatrix}$$

Find the mean of these rows,

$$\overline{v_{x(t)}} = \frac{1}{10}\sum_{i=0}^{9} v_{x(t-i)}$$

$$\overline{v_{y(t)}} = \frac{1}{10}\sum_{i=0}^{9} v_{y(t-i)}$$

Then find the difference matrix,

$$D_{(t)} = V_{(t)} - \overline{V_{(t)}} = \begin{pmatrix} \mathrm{v}_{x(t)} - \overline{v_{x(t)}} & \mathrm{v}_{y(t)} - \overline{v_{y(t)}} \\ \mathrm{v}_{x(t-1)} - \overline{v_{x(t)}} & \mathrm{v}_{y(t-1)} - \overline{v_{y(t)}} \\ \mathrm{v}_{x(t-2)} - \overline{v_{x(t)}} & \mathrm{v}_{y(t-2)} - \overline{v_{y(t)}} \\ .... & ... \\ ... & ..... \\ \mathrm{v}_{x(t-9)} - \overline{v_{x(t)}} & \mathrm{v}_{y(t-9)} - \overline{v_{y(t)}} \end{pmatrix}$$

The covariance matrix $C_{(t)}$ is calculated by,

$$C_{(t)} = D^T_{(t)}.D_{(t)}$$

$C_{(t)}$ is decomposed into
$C_{(t)} = P_{(t)}D_{(t)}P^{-1}_{(t)}$ using eigen value decomposition.

$$P_{(t)} = \begin{pmatrix} \mathrm{w}_{1x(t)} & \mathrm{w}_{2x(t)} \\ \mathrm{w}_{1y(t)} & \mathrm{w}_{2y(t)} \end{pmatrix}$$

and

$$D_{(t)} = \begin{pmatrix} \lambda_{1(t)} & 0 \\ 0 & \lambda_{2(t)} \end{pmatrix}$$

Here the $P_{(t)}$ has the eigen vectors,

$$\underline{w}_{1(t)} = w_{1x(t)}\underline{i} + w_{1y(t)}\underline{j}$$

$$\underline{w}_{2(t)} = w_{2x(t)}\underline{i} + w_{2y(t)}\underline{j}$$

$D_{(t)}$ has their corresponding eigen values $\lambda_{1(t)}$ and $\lambda_{2(t)}$

The bigger value of $\lambda_{1(t)}$ and $\lambda_{2(t)}$ is chosen (let it be $\lambda_{1(t)}$) and the corresponsing eigen vector $\underline{w}_{1(t)}$ gives the direction of the breathing.

The unit vector along this direction is calculated for the next steps,

$$u(t) = \frac{\underline{w}_{1(t)}}{\|\underline{w}_{1(t)}\|}$$

- Now we have $\underline{u}(t)$ and $\underline{v}_{(t)}$. Projecting the velocity vector in the unit vector of direction gives a scalar parameter $s_{0(t)}$ that can be used to determine breathing.

$$s_{0(t)} = \underline{u}(t).\underline{v}_{(t)}$$

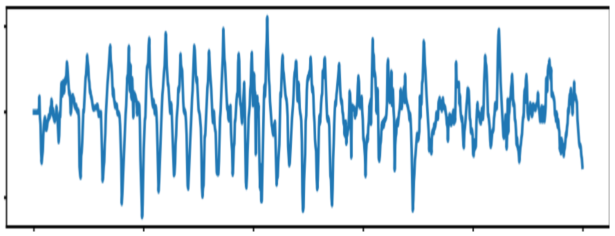

Figure 3.6: $s_{0(t)}$

- $s_{(0)t}$ undergoes two smoothing techniques to give,

$$s_{1(t)} \leftarrow \text{ adaptive filtered } s_{0(t)}$$

$$s_{2(t)} \leftarrow \text{ gaussian blurred } s_{1(t)}$$

$$s_{1(t)} = s_{0(t)} \times 0.8 + s_{0(t-1)} \times 0.2$$

Gaussian blurring is done with 15σ radius.

- The graph $s_{2(t)}$ - t looks as follows,

- The peaks are found using the technique,

$$s_{2(t)} > s_{2(t-1)} \text{ and } s_{2(t)} \leq s_{2(t+1)} \Rightarrow s_{2(t)} \text{ is a peak.}$$

The peaks are marked as following.

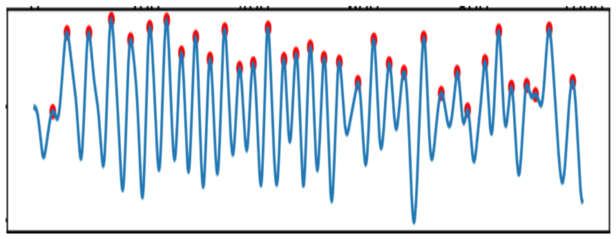

Figure 3.7: $s_{2(t)}$ with peaks marked

- The breathing intervals are calculated as the time between two peaks.

Breath number	Time for cycle / (ms)
1	1200
2	1230
3	1050
4	1050
...	...
...	...
...	...
...	...

Table 3.1: Breathing time interval report

3.1.8 Communication

The system uses MQTT protocol to communicate.
This was implemented using paho-MQTT library by eclipse.
The client publishes to 3 subtopics under the main topic : *user_name/device_ID*.
The three subtopics are :

- Data
  The device publishes the actual data to this topic. The data is sent as a json message with two parameters : Timestamp, Array of data. The timestamp helps in identifying duplicates and ordering. The array of data is used to plot the graph

- Connection
  This is the last will topic of a device. When a device disconnects (expected or unexpected manner) the broker identifies this and publishes a disconnect message to the topic. Therefore when the device connects it publishes 1 to this topic to denote that it is active and when it disconnects, the broker

publishes 0 to this topic to denote that it is inactive (This is used in the Web interface).

- Alert
  The algorithm provides timely alerts when there is a possibility of sleep apnea. The device publishes to this topics in case such an alert occurs. (This feature has been disabled due to the high number of false positives. To enable please change the flag in config.in to true).

3.1.9 Security

The communication between Node and the Broker was secured using multiple techniques. All these features are provided by paho-MQTT library. The security features are actually implemented in the Broker but the device must be configured to use them. Read the security section under broker for further details.

- Authenticate
  Each device has device ID and access token. By combining the device ID with the user name each device would have a unique ID (Different user could have device with same device ID therefore user name and device ID are combined). The combined ID acts as the MQTT user name and the access token acts as the password. These two are used to authenticate the device.
  The MQTT user name and password are stored in the config.ini file and is read using *read* function in the *configparser* library.
  Then the user name and password can be set using *username_pw_set* function in the MQTT library.

- Access Control
  Each device has a particular topic to which it could publish. The device doesn't have permission to publish to any other topics unless the device ID and the access token are changed. The topic to which the device can publish is: *user_name/device_ID* and the device can publish to any subtopic under it.

- Encryption
  The communication uses TLS encryption to communicate with the Broker. Since the authentication send the password this is a must. It also ensures confidential transmission of data.
  No application level encryption is used since the MQTT model uses multiple subscriber publisher model and this would make key exchange mush more complicated. Multiple subscribers imply that all the clients would need to know the key which makes it impractical.
  TLS certificate was obtained from the broker and can be set using *tls_set* function in the MQTT library.

3.1.10 Testing

Each segmant was initialy tested as a seperate unit and finaly they were connected together and tested. The information regarding the tests are as follows:

Camera Alignment

1. First, the functionality of the camera is tested. Video feed taken using the camera was communicated to a computer using a communication protocol and the received video was displayed in computer screen. The color, sharpness, intensity of the camera module was tested in order to configure correct parameters.

2. Localization of the abdomen of the infant was tested using prerecorded videos. They were analyzed in a computer and results were obtained from three different types of videos. Results are shown below. White mask is the predicted abdomen area, using filtering techniques and connected component analysis.

3. The correct rotation of the camera module is tested using a picture of the infant and gently moving the image. It is verified that the camera follows the image with given configurations. *Note*: This algorithm cannot identify sudden movements. But it won't be a problem since infants cannot move their body quickly.

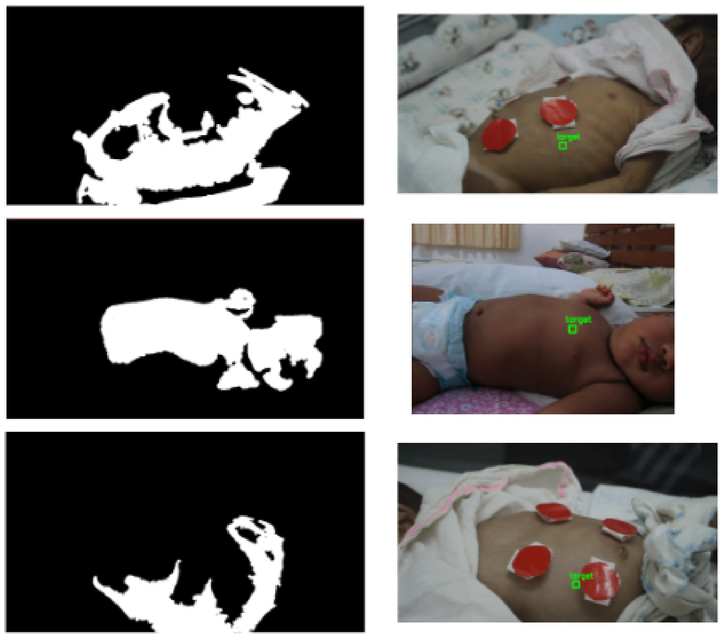

Figure 3.8: Baby detection algorithm

Breathing pattern detection

- Since the video data set was not labelled, the testing required labelling as well.
- A user friendly interface for video frame labelling was developed.
- The accuracy of the algorithm was tested by comparing the results of the algorithm to the human labelled data.
- code : https://github.com/TeambiteCode/Sleep_Apnea_Detection/tree/master/Testing
- The results of the testing were used to tune the parameters of the algorithm such as
  - Radius of Gaussian filters

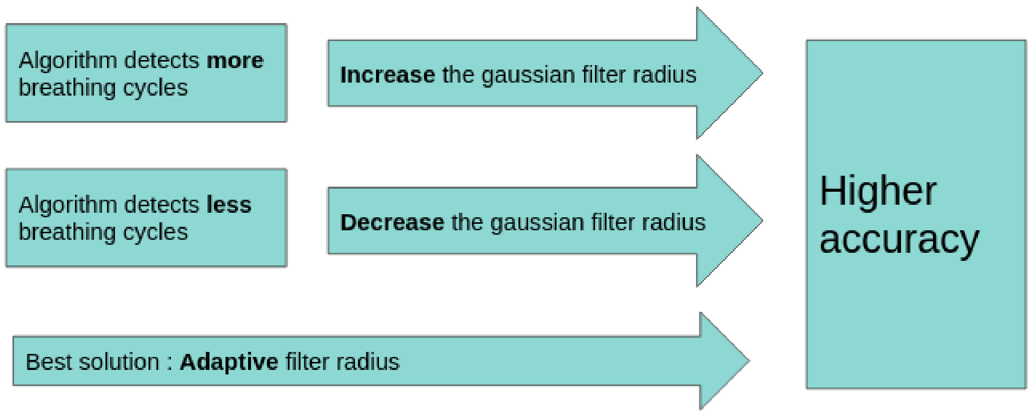
Algorithm detects **more** breathing cycles
Increase the gaussian filter radius
Algorithm detects **less** breathing cycles
Decrease the gaussian filter radius
Best solution : **Adaptive** filter radius
Higher accuracy

Figure 3.9: Unit testing of the algorithm to improve accuracy

MQTT Communication

Since Communication cannot be tested as a seperate component, it was tested together with the broker. Refer Testing Under Broker.

Integration testing

The full system has been only tested with data sets since we are still awaiting a response from the medical faculty to test the equipment.

3.2 MQTT Broker

MQTT plays a huge role in this system since all the real time data are send directly to the user instead of using the server. MQTT uses a publisher-subscriber mechanism (as opposed to client-server). This ensures that single client can write/listen to multiple topics while multiple clients can write/listen to the same topic. This is a form of m-n communication and, if applied in a client-server model could be slow and inefficient. To do this efficiently and in real time, MQTT was chosen. MQTT messages are much more smaller (small headers) thus offer high throughput, but cannot support larger messages. In our case this is not a problem since messages sent are small is size.
Code for broker implementation can be obtained from:
https://github.com/TeambiteCode/Sleep_Apnea_Detection/tree/master/Mosquitto.

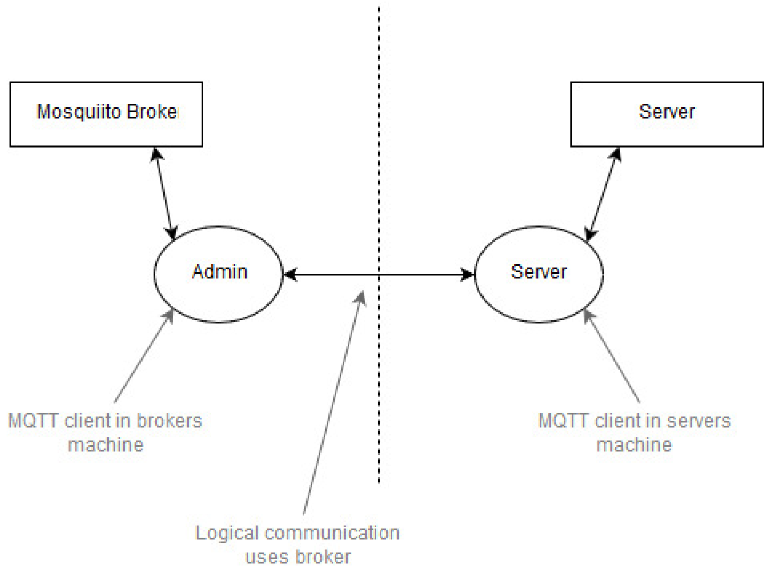
Mosquiito Broke
Server
Admin
Server
MQTT client in brokers machine
MQTT client in servers machine
Logical communication uses broker

Figure 3.10: Server broker architecture

The MQTT broker was implemented independent to the server. This made testing easier, system more secure (Breach in one system would keep the other intact) and gave much more flexibility while developing specific features. This would also allow us to deploy them on different servers.
Therefore the Broker must communicate with the server through a separate mechanism. In our case two MQTT clients with the following names were used:

- Admin - Runs with the broker. Controls Broker
- Server - Runs with the client. Communicates Broker

This is the only pathway through which the broker and server communicate and transfer information. It is used when signing in or registering new device.The *Server* publishes to the Admin topic to which the *Admin* listens and the *Admin* publishes to the server topic to which the *Server* listens. Since this is a one to one communication it uses AES algorithm to encrypt the data.

The implementation of MQTT client *Admin* is explained in this section and the implementation of MQTT client *Server* is explained in the next section

3.2.1 Mosquitto

Mosquitto is used as the MQTT Broker. It can be installed as follows:

```
sudo apt-get install mosquitto
```

The MQTT broker listens to 3 ports :

- 1883 (MQTT without TLS)
- 8883 (MQTT with TLS)
- 9001 (Websocket)

The listeners can be opened using *listener* in Mosquitto config file (mosquitto.conf). The protocol of each listener must be specified. All other configuration are available in the default mosquitto config file. Configuring the security features are explained in the next section.

3.2.2 Admin

The admin client is much similar to the MQTT client in the device. It is a python based client created using paho-MQTT library. Only the differences are explained in this section.
Code for admin implementation can be obtained from:
https://github.com/TeambiteCode/Sleep_Apnea_Detection/tree/master/Mosquitto/Admin.

- The client has a user name ("Admin") and password previously set and these are specified in static files (Static files explained under security/authentication).
- The admin is subscribed to the topic . This is the global wild card and as a result the admin gets all the messages. Since admin is a superuser it has access to all the topics. Thus the Admin can communicate with both the server and the devices.
  The subscription to this particular topics can be done using the *subscribe*

function in the MQTT library. The admin publishes to the *Server* topic. This is done using the publish function.

- Admin controls the MySQL database used by Broker and connection to this is made using MySQLdb connector library for python.
- In addition to TLS protection the admin uses application level encryption. To implement this, AES algorithm and Diffie-Hellman Key Exchange protocol is used. (Explained under security). Authentication and acl are also explained in security.

The *Admin* MQTT-client provides the following functionality.

Register Device

When a device is registered the *Server* MQTT client sends a message to this topic with the user name and password (encrypted). Admin decrypts it and then hashes the password before adding it into the user table.

User login

All real time updates are sent directly from the MQTT broker (Not the server) thus the user needs access to these topics. When a user logs in *Server* (MQTT client) authenticates the user and sends a message to this topic with the user name and password (encrypted). Admin adds a temporary entry in the user table with the user name and creates a temporary access token for the user. Then in the acls table the *Admin* gives access to these topics to the user.

Then every time the Server requests for a config.js file, it checks database and creates a config.js file with the user name, password, topic, host, port, etc. and send it to the *Server* (MQTT Client) by publishing it to the topic "Server" .When the server requests the config file, if the user name is not present in the table it doesn't add a new entry but instead sends the config.js file with no user name, thus the client would get a 404 error.
The admin ensures that these entries are temporary using 4 flags:

1. last will - When a client disconnects a message is posted to the *disconnect* topic by the broker. This is the last will topic. The admin listens to this and removes the entry when the user disconnects. Disconnection occurs if user logs out or closes tab.
2. keep-alive - When the client swiches from onee page to another the connection closes, but the entry should not be deleted (since he is still logged in). In this case a 300s keep-alive condition is configured in client so the broker doesn't publish to last will topic even if the client disconnects. Once the client reconnects the broker identifies this and keeps the connection alive.

3. persistence_limit- If the temporary entry exists indefinitely it is dangerous. Thus a persistence limit of 1hr is configured. The broker disconnects the connection after this time period and posts to the last will topic. The admin deletes the entry. The client tries to reconnect when disconnected and as a result a new connection with the broker is made using a new temporary entry.

4. QOS - During the disconnection some information would be lost. In order to avoid this, a QOS of 1 is set for each client. As a result the broker would store the last message to each topic and when a new user connects it would send that message.

3.2.3 Security

The broker implements security at multiple levels. At the network level a firewall is used. At transport level it uses TLS encryption. Apart from this authentication and access control in done using auth plugin by jpmens. Communication between broker and server is done using a MQTT client at each end and the communication between these 2 clients are encrypted using AES. Diffie-Hellman Key Exchange protocol is used for key exchange. Since sensitive information such as passwords are interchanged this encryption is a must.

- Firewall
  The firewall blocks all other ports except for 1883 (MQTT without TLS) 8883 (MQTT with TLS) and 9001 (Websocket). Firewall was implemented using ufw (Uncomplicated Firewall) in linux (Mosquitto runs on Linux).
  Firewall can be enabled using *ufw enable* command
  All incoming and outgoing traffic could be blocked using *ufw default deny incoming* and *ufw default deny outgoing* commands
  The particular ports were allowed using *ufw allow <port_no>* command.

- Authenticate
  Authentication is done using mosquitto auth plugin with MySQL backend.

  1. Mosquitto auth plugin can be obtained from github,
  2. Open config.mk and change both *files* and *mysql* to yes and then make
  3. Next open a blank text file and add the static user name and passwords (In our case the admin client and server client
     admin : admin
     server : server
  4. Convert to password file using following command:
     mosquitto_passwd - c filename

5. Include this in the mosquitto.conf file using auth_opt_password_file option.
6. Create a database in mysql with table "*users*" with columns "*id*" "*username*", "*password*" and "*super*". Each time a device is registered, the username and hashed password would be added to this database by *Admin* MQTT client. Each time a user logs in, *Admin* does the same thing.
7. Finally open mosquitto.conf file and include the auth plugin. This includes setting the auth_plugin, auth_opt_backend, host, port, user, pass, etc.

- Access Control
  Access control is done in a hierarchical manner:

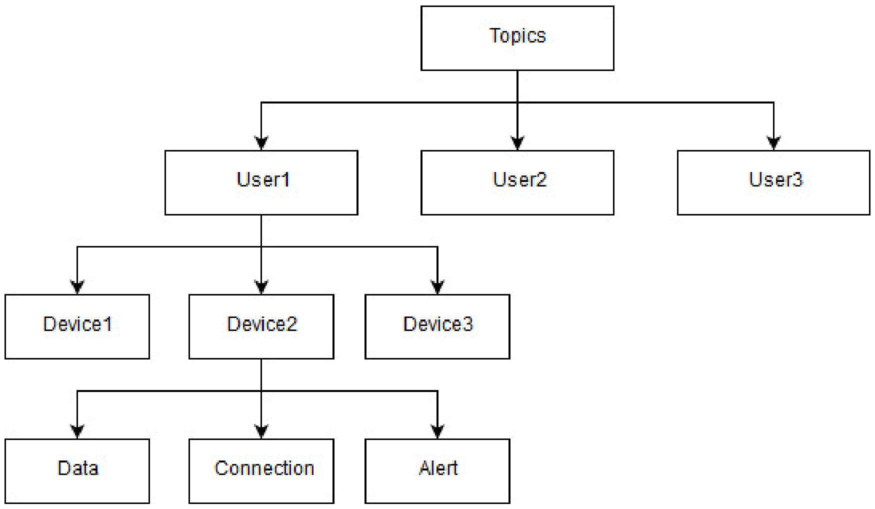

Figure 3.11: ACL

A user has read access to all the device topics, but the device has read write access to only its device topic as shown in the diagram.

ACL is aslo implemented using mosquitto auth plugin.

1. Open the same database in MySQL as in authentication and create a new table "*acls*" with columns "*id*", "*username*", "*topic*" and "*rw*". Each time a new device is created *admin* (MQTT client) adds it to acl with the corresponding topic. Each time a user logs in *admin* adds a temporary entry.
2. Since the configuration in the conf file has been done in the previous step no need to repeat.

- Encryption
  The communication uses TLS encryption to communicate with the devices and users. It was done using the following steps.

  1. A new certificate was created as follows:
     ```
     openssl genrsa -out server.key 2048
     openssl req -new -out server.csr -key server.key
     ```

  2. The certifice was self singed (It can be signed using a CA as well) as follows:
     ```
     openssl genrsa -des3 -out ca.key 2048
     openssl req -new -x509 -days 1826 -key ca.key -out ca.crt
     openssl x509 -req -in server.csr -CA ca.crt -CAkey ca.key -CAcreateserial
     -out server.crt -days 360
     ```

  3. The generated certificate was added to the broker and the mosquitto.conf file was configured to include it.
  4. The public certificate was added to the device.

  Apart from this the communication between Admin and Client uses AES encryption and DH key exchange. AES encyption is implemented using cryptodome package and DK key exchange is implemented using pyDH package. Both these packages can be installed using pip. The steps are as follows:

  1. Install necessary packages as follows.
     ```
     pip install cryptodome
     pip install pyDH
     ```
  2. The key exchange is done using *DiffieHellman()* and *gen_public_key()* functions and the shared key is generated using *gen_shared_key()* function in the pyDH library.
  3. Now that we have the shared key, the key is converted to AES usable key using AES.new() function and the encryption and decryption are done using encrypt_and_digest() function and decrypt_and_verify() function respectively. Note that the funtion creats both the cipher text and the tag where the tag is used for verification.

Finally the broker could be run.
To start database :

sudo service mysql start

To start Broker :

mosquitto -c mosquitto.conf

3.2.4 Testing

Code for broker testing can be obtained from:
https://github.com/TeambiteCode/Sleep_Apnea_Detection/tree/master/Mosquitto/Test.

Mosquitto provides mosquitto_pub and msoquitto_sub libraries for testing purposes but these were not used since their usage is limited. Instead MQTTlens was used to test which gives much interactive way to visualize the output.

There are no standard tests for testing mosquitto thus, the method used was based on trial and error. Multiple test cases(python clients) were created. This included clients with different user names, passwords, topics, anonymous clients (without setting username or password). On the other hand the tests were verified by using the mosquitto log and MQTTlens.

The log was redirected to "stdout" by setting the log_dest in the mosquitto.conf file. The connections, disconnections and failures were observed through the log. This was used to verify that the user had to use the correct user name and password, and the TLS file had to be specified if connecting to specific port. Anonymous users were blocked on specific ports, and all other connections to unspecified ports failed. (When wrong user name or password is given the connection is refused, so the client disconnects, but if the client tries to connect to unspecified port, the firewall blocks the connection and as a result the connection fails and an error message would be displayed on the clients side. The only successful connections were, when both the user name and password were present in the mosquitto back end database. Therefore it was confirmed that the firewall and the authentication mechanism were functional.

The MQTTlens was subscribed to the topic (multilevel) wild card with the user name "admin" and password "admin". Since admin is a super user, it could listen to all the topics. The test for access control was done using this. The mosquitto log would not display even if a client tried to publish / subscribe to an unauthorized topic. But the publish/ subscribe mechanism could be checked using a super user with access to all topics. Successfully connected clients from the previous stage were used to publish and subscribe to multiple topics. While the admin received all the messages, only the authorized clients, were able to publish / subscribe to particular topics. ie: The admin only received messages published by clients with access to particular topics. Therefore it was verified that the ACL mechanism was functional as well.

3.3 Server

The server-side was designed using python based django and rest-framework. As of djnago initially to facilitate the production environment it comes with a light weight django server and a SQLlite database. Thus the production was initally done using django's server. Since it is a lightweight production level server in order to facilitate a better deployment value the project was later transferred to an Apache server in deployment. Since the project required a site with separate clients with critical data it became necessary to build a reliable user login and session management system. So django's default login schema which facilitated session based user login system was used.

Code for Server can be obtained from:
https://github.com/TeambiteCode/Sleep_Apnea_Detection/tree/master/Server.

The functionality provided by django is explained below:

3.3.1 Signup

For the signup part each attributes where obtained separately and were obtained and processed to validated for existence and then where assigned to django's basic authentication models. When sending passwords via a the post method to avoid the theft of then sha256 algorithm was used with 100000 iterations and the final hash was stored in the server for further authentication purposes. Rest of the data where sent in a normal manner so that it can facilitate a faster access when needed sice they are not a one time used attribute in a session.

3.3.2 Login

When logging in the same strategy of obtaining the password and sending the hash to the server for further processing was used for security purposes. Once the login is facilitated relevant user was saved in the session so that the user can access the data without having to signing in every single time.

3.3.3 Register device

In the model a user was allowed to hold several devices. Thus in the schema devices are created with the users as a primary key. Addition devices where facilitated by the use of two particular ways. One was via the interface. Second one was with a json request to the rest api along with the user authentication(since a browser is not used). The devices were also responsible for the addition of data. Since we have to deal with a million data points it made sense to break the data into easily process able chunks. Thus data points were primary keyed with the time stamps associated with them and the respective devices. The data insertion was facilitated via a json response through the rest api thus enabling the devices to automate the process rather than someone inserting via an interface all the time. The data can also be retrieved via the time stamps facilitating a better data transportation. And both the time data and the device data were denied even the read only access for an un authorized party.

3.3.4 Delete device

The deletion of devices similar to the registering were facilitated via both the interface and json response thus giving the user the option of an automated device management if the need arises. The device related data was set to cascade on device deletion thus saving memory. Apart from them via a json response with the authentication the ability to delete data with time stamps were also used.

3.3.5 Logout

Logout process was done by just removing the user details saved in the session data.

3.3.6 Security

Improper implementation of the database can lead to a sql injection problem. By the use of django's default user model a default check on sql requests were made manually to prevent the improper access to the sql database. At the meanwhile a malicious user can do a Cross-Site Request Forgery (CSRF) by the use of another page to send requests on behalf of the user to obtain sensible data. So when doing the post requests a CSFR token which is a non guessable and protected by HTTP's access control random number generated by the server is added to the forms. Thus the use of this large token enabled only the right client to make requests on their behalf. And as far as password transaction via a HTTP is concerned to protect it and sha256 hashing was used as mentioned before. And for the rest api's since the data is sensible an authentication only access schema was set thus even denying the read only access to the non authenticated users.

3.3.7 MQTT Client

As explained in the previous section a MQTT client with the user name "*server*" runs on the server side. This is used to communicate with the Broker. Since the broker doesn't actually listen, the communication is done with another client in the broker side (Admin) and in turns admin does the necessary changes in broker. The Server publishes to the topic admin *Server* to which the Admin listens and the Admin publishes to the topic server to which the Server is subscribed. The publish and subscribe would be referred as send and receive.

For security the MQTT client uses pyDH and cryptodome. These sections were explained earlier (The implementation is the same. Refer Admin, Security in MQTT broker for implementation)

"Server" MQTT client provides two functions:

Register Device

When a device is registered the broker should be notified
MQTT client sends a message to Admin with the user name and password (encrypted). The publishing is not done like any previous methods (creating a client then publishing) but instead it is done using a single line helper function. The helper function (Also provided by paho-MQTT library) is a one line implementation, which is used when a single message needs to be sent or received. The implementation is as follows:

```
single("admin", payload=client.json, qos=1, hostname="<MQTT_host>",
    port=8883, auth={'username':"Server", 'password':"server"},
    tls = {'ca_certs':"<ca_certs>}, protocol=mqtt.MQTTv311, transport="tcp")
```

The single function is available in the paho.mqtt.publish library. Here the first argument admin is the topic and the payload is the encrypted username and password. Apart from this, the host and the tls will have to be specified. A message received authentication is not needed since a QOS of 1 is set (sent atleast once).

User login

When a user logs in the Django does the authentication. If successful, *Server* (MQTT client) gets the user and sends a message to the Admin with the user name and password (encrypted).

Then every time the user requests for the config.js file, the Server (MQTT Client) sends a message to the Admin with the user name and the topic needed. Then the Server waits for the reply from the Admin. Once received, this file is sent to the user. Therefore instead of dynamically creating this at the Server, the config file is made by the broker (The way the admin handles this requests is explained in the admin section).
The sending message is done using the helper function as mentioned in the previous topic. The receiving message is also done as follows.(The name of function is simple):

```
msg = simple("server", qos=0, msg_count=1, retained=False,
    hostname="<MQTT_host>", port=8883,
    auth={'username':"Server", 'password':"server"}, tls = {'ca_certs':"<ca_certs>},
    protocol=mqtt.MQTTv311, transport="tcp")
```

The simple function is available in the paho.mqtt.subscribe library. Notice that retained is False. This is important to make sure that retained messages are not received. Apart from this the hostname and tls needs to be set like the previous method. Once received, this message is sent to the user.

These actions should be trigerred when the user requests for the config.js file. This is done by making a view for this particular URL in django and including these functions inside the view. When a config.js file is requested, the server sends a request to admin and the admin replies the config.js file which is then redirected to the user.

3.3.8 Testing

The server is made with certain schema for models and the database. But when it came to testing we had to check about their perfection. So tests were necessary. But running a buggy test on the main database may ruin the structure and the data in it. Thus for the testing purpose a new database was formed, a copy of the original one with the test period as it's lifetime and the unit tests where done on them. First the test was done for the creation of objects of the database model. Then the url mappings and rendering where validated along with the views that managed the mapping. When the tests where done they where done divided by their functionality rather than a whole. This unit testing mechanism helped us identify the bugs easier and faster.

3.4 Web Interface

The Web interface plays a huge part, since most of the dynamic contents are generated at the users machine using MQTT client rather than enforcing it from the server. Therefore the workload on the server would be reduced whereas the workload on the broker would increase.
Note that the MQTT broker uses publish subscriber mechanism therefore the incremental workload is negligible.
Also Note that the usage of users resources would be simillar (Server Push vs MQTT).
The Web interface was developed using HTML, CSS , javascript.
Apart from these the following libraries and plugins were used:

- Bootstrap
- MetisMenu
- DataTables
- Flot
- Eclipse Paho JavaScript client

The web interface consist st of multiple components. Each component of the Web Interface is described below

3.4.1 Navigation Bar

The navigation bar is included in all the pages. The navigation has two section the top bar and side bar.

- Top bar
  The top bar provides two feature: logout and alerts. The alert is done using MQTT javascrpit client. The client obtains the config.js file from the server. It mentions the topic as <username>/+/alert (The + is a single topic wild card therefore any alerts from devices belonging to that particular user will be listened by this client). Therefore whenever the device sends an alert, the web interface displays it by adding it to the list.
  The logout functionality is implemented in the server (Refer server).

- Side bar
  The side bar is used to go to multiple pages. It uses the metis menu package which makes the menus much more interactive.

3.4.2 Dashboard

The dashboard provides alerts as to when a device is connected or removed. It uses Bootstrap alerts div, to show the alerts but the functionality is implemented using a MQTT javascript client. A request is made to the server and it returns the config.js file. It specifies two topics : <username>/+/alert and <username>/+/connection. Therefore whenever a device belonging to the particular user connects/ disconnects or whenever an alert is sent it would be notified to the user. The MQTT message is read and a alert is created and added to the dashboard list.

3.4.3 Add Device

The add device uses a form. The devices can be added using this form. The implementation is explained in the server.

3.4.4 Device List

Device list uses DataTalbe which provides a more interactive way to visualize tables. The population of the table is done in the server. Apart from this it provides a delete device functionality which is also implemented in the server.
The table has a specific column known as active which is used to see if the device is active. This is done using MQTT javascript client. The client subscribes to <username>/+/connections topic. Whenever a device connects a disconnects, a message is published to this topic. Depending on the message and the device ID the active value corresponding to a device is changed to green (active) or red

(inactive). (Here the "+" is a wild card for and device_id and by splitting the topic, the 2nd part of the topic would give the device ID)

3.4.5 Charts

The charts is done through flot. This provides a way to plot real time moving charts.
The data for the charts is obtained using MQTT javascript client. The client subscribes to <username>/<device_id>/data topic. When the message is received, it pushes(enque) the data to the existing queue and plots the queue. If the queue overflows that additional length is removed (dequed).

3.5 Downloading the software

The software for the node device and the server (in case you want to run your own server without using **http://isad.teambitecode.com**) is released as open-sorce code in **https://github.com/TeambiteCode/Sleep_Apnea_Detection**
The latest release as of now is v1 availiable at
https://github.com/TeambiteCode/Sleep_Apnea_Detection/tree/master/releases/v1.
For the ease of use, tarballs of the code is released here

- http://teambitecode.com/projects/sleep-apnea-detection-node.tar
- http://teambitecode.com/projects/sleep-apnea-detection-server.tar

Following are the instructions on getting, setting and running the code.

3.6 Device

```
wget http://teambitecode.com/projects/sleep-apnea-detection/node.tar
tar -xvf node.tar
cd node
export $SDCARD=[The mount location of the SD card here]
export $WIFI=[Wifi network ssid name here]
export $WIFIUSER=[Wifi username here]
export $WIFIPASS=[Wifi password here]
sudo chmod 700 install.sh
sudo ./install.sh
```

3.7 Server - webserver and MQTT broker

```
sudo apt-get update
sudo apt-get install apache2
sudo apt-get install mysql-server
sudo apt-get install php libapache2-mod-php php-mcrypt php-mysql
wget http://teambitecode.com/projects/sleep-apnea-detection/server.tar
tar -xvf server.tar
cd server
sudo chmod 700 install.sh
sudo ./install.sh
```

END