Embedded System for Chronic Disease Patient Monitoring using IoT

Generated by Doxygen 1.9.1

Author: Mr. Mohamed Ali Haoufa

Documentation version: 1.0

Date: August 17th, 2023

Chapter 1

Embedded-System-for-Chronic-Disease-Patient-↩ Monitoring-using-IoT

This project aims to develop an embedded system that monitors chronic disease patients who need frequent medical check-ups.

This embedded system uses a Raspberry Pi 4b to gather data from several sensors installed in patients' bodies, and their data is constantly saved, analyzed and processed to extract vital details about their health and transfer it safely using MQTT protocol with the help of security encryption (AES Algorithm) to the IoT cloud where it is organized into an interface making it easier to use and understand, which allows users like doctors to easily access patients' data and observing their health state from any place in the world.

To enhance the functionality of the system, a warning system (Email/SMS) has been integrated into the IoT platform, which warns the doctor when the patient enters an abnormal state.

The budget and the outcome of this project are reasonable compared to what the patients pay for their Regular medical check-ups.

Keywords: Embedded system, Chronic Disease, Health, patients, MQTT protocol, Security Encryption, AES (Advanced Encryption Standard) Algorithm, IoT Platform, Sensors.

Chapter 2

File Index

2.1 File List

Here is a list of all documented files with brief descriptions:

Chapter 3

File Documentation

3.1 src/ECG_Noise_Filter.py File Reference

In this sub-program, we have combined multiple filters to create our final filter.

Functions

- def ECG_Noise_Filter.butter_highpass (cutoff, fs, order=5)

  Apply a Butterworth high-pass filter to a signal.
- def ECG_Noise_Filter.butter_lowpass (cutoff, fs, order=5)

  Apply a Butterworth low-pass filter to a signal.
- def ECG_Noise_Filter.notch_filter (cutoff, q, fs)

  Apply a notch filter to an ECG signal.
- def ECG_Noise_Filter.highpass (data, fs, order=5)

  Apply a high-pass filter to input data.
- def ECG_Noise_Filter.lowpass (data, fs, order=5)

  Apply a low-pass filter to input data.
- def ECG_Noise_Filter.notch (data, powerline, q)

  Apply a notch filter to remove powerline interference from input data.
- def ECG_Noise_Filter.final_filter (data, fs, order=5)

  Apply a series of filters to the input data for noise reduction and signal enhancement.

Variables

- int **ECG_Noise_Filter.cutoff_high** = 20
- int **ECG_Noise_Filter.cutoff_low** = 600
- int **ECG_Noise_Filter.powerline** = 60

3.1.1 Detailed Description

In this sub-program, we have combined multiple filters to create our final filter.

Author

- Mr. Mohamed Ali Haoufa
- Mr. Mohamed Nacer Namane

Date

Jun 7, 2022

Copyright

3.1.2 Function Documentation

3.1.2.1 butter_highpass()

```
def ECG_Noise_Filter.butter_highpass (
              cutoff,
              fs,
              order = 5 )
```

Apply a Butterworth high-pass filter to a signal.

This function applies a high-pass filter to a signal to attenuate frequencies lower than the specified cutoff frequency. The order of the filter determines the roll-off rate of the attenuation. The resulting filter coefficients (b, a) can be used to filter the signal using the `scipy.signal.lfilter` function.

Parameters

cutoff	The frequency below which lower frequencies will be attenuated, in Hertz.
fs	The sampling frequency of the signal in Hertz.
order	The order of the Butterworth filter (default is 5).

Returns

: The filter coefficients (b, a) for the high-pass filter.

3.1.2.2 butter_lowpass()

```
def ECG_Noise_Filter.butter_lowpass (
              cutoff,
              fs,
              order = 5 )
```

Apply a Butterworth low-pass filter to a signal.

This function applies a low-pass filter to a signal to attenuate frequencies higher than the specified cutoff frequency. The order of the filter determines the roll-off rate of the attenuation. The resulting filter coefficients (b, a) can be used to filter the signal using the `scipy.signal.lfilter` function.

Parameters

cutoff	The frequency above which higher frequencies will be attenuated, in Hertz.
fs	The sampling frequency of the signal in Hertz.
order	The order of the Butterworth filter (default is 5).

Returns

: The filter coefficients (b, a) for the low-pass filter.

3.1.2.3 final_filter()

```
def ECG_Noise_Filter.final_filter (
              data,
              fs,
              order = 5 )
```

Apply a series of filters to the input data for noise reduction and signal enhancement.

This function applies a series of filters in sequence to the input data in order to achieve noise reduction and enhance the signal. The process includes a high-pass filter, a low-pass filter, and a notch filter to remove unwanted frequency components.

Parameters

data	The input signal data to be filtered.
fs	The sampling frequency of the input data, in Hertz.
order	The order of the Butterworth filters used for high-pass and low-pass filtering.

Returns

: The filtered and enhanced signal after applying the series of filters.

3.1.2.4 highpass()

```
def ECG_Noise_Filter.highpass (
              data,
              fs,
              order = 5 )
```

Apply a high-pass filter to input data.

This function applies a high-pass filter to the input data using a Butterworth high-pass filter with the specified order. The resulting filtered signal is returned.

Parameters

data	The input signal data to be filtered.
fs	The sampling frequency of the input signal, in Hertz.
order	The order of the Butterworth high-pass filter (default is 5).

Returns

: The filtered signal after applying the high-pass filter.

3.1.2.5 lowpass()

```
def ECG_Noise_Filter.lowpass (
              data,
              fs,
              order = 5 )
```

Apply a low-pass filter to input data.

This function applies a low-pass filter to the input data using a Butterworth low-pass filter with the specified order. The resulting filtered signal is returned.

Parameters

data	The input signal data to be filtered.
fs	The sampling frequency of the input signal, in Hertz.
order	The order of the Butterworth low-pass filter (default is 5).

Returns

: The filtered signal after applying the low-pass filter.

3.1.2.6 notch()

```
def ECG_Noise_Filter.notch (
              data,
              powerline,
              q )
```

Apply a notch filter to remove powerline interference from input data.

This function applies a notch filter to the input data in order to remove powerline interference at a specific frequency. The resulting filtered signal is returned.

Parameters

data	The input signal data to be filtered.
powerline	The frequency of the powerline interference to be removed, in Hertz.
q	The quality factor (Q) of the notch filter.

Returns

: The filtered signal after applying the notch filter.

3.1.2.7 notch_filter()

```
def ECG_Noise_Filter.notch_filter (
              cutoff,
              q,
              fs )
```

Apply a notch filter to an ECG signal.

This function applies a notch filter to an ECG signal to attenuate frequencies around the specified cutoff frequency. The quality factor (q) determines the width of the notch. The resulting filter coefficients (b, a) can be used to filter the signal using the `scipy.signal.lfilter` function.

Parameters

cutoff	The frequency to be attenuated in Hertz.
q	The quality factor of the notch filter.
fs	The sampling frequency of the ECG signal in Hertz.

Returns

: The filter coefficients (b, a) for the notch filter.

3.2 src/IOT_Change_Send_Functions.py File Reference

In this sub-program, we developed certain functions to encrypt, change and send the collected data to the IoT platform.

Functions

- def IOT_Change_Send_Functions.encrypt (data, key, iv)

  Encrypts the given data using AES encryption.
- def IOT_Change_Send_Functions.MAX_value (parameter, number)

  Calculates the maximum value by adding a number to the parameter.
- def IOT_Change_Send_Functions.MIN_value (parameter, number)

  Calculates the minimum value by subtracting a number from the parameter.
- def IOT_Change_Send_Functions.Temp_Sending_Condition (temp, temp_min, temp_max, temp_data)

  Checks if the temperature value is within the specified range before sending.
- def IOT_Change_Send_Functions.BPM_Sending_Condition (BPM, BPM_min, BPM_max, BPM_data)

  Checks if the BPM value is within the specified range before sending.
- def IOT_Change_Send_Functions.IBI_Sending_Condition (IBI, IBI_min, IBI_max, IBI_data)

  Checks if the IBI value is within the specified range before sending.
- def IOT_Change_Send_Functions.RMSSD_Sending_Condition (RMSSD, RMSSD_min, RMSSD_max, RMSSD_data)

  Checks if the RMSSD value is within the specified range before sending.
- def IOT_Change_Send_Functions.cvt_2_base64 (filename)

  Converts an image file to a Base64-encoded string.
- def IOT_Change_Send_Functions.image_generator (data)

  Generates and saves an image of the ECG signal.

Variables

- int **IOT_Change_Send_Functions.INTERVAL** = 0
- int **IOT_Change_Send_Functions.temp_min** = 0
- int **IOT_Change_Send_Functions.temp_max** = 0
- int **IOT_Change_Send_Functions.BPM_min** = 0
- int **IOT_Change_Send_Functions.BPM_max** = 0
- int **IOT_Change_Send_Functions.IBI_min** = 0
- int **IOT_Change_Send_Functions.IBI_max** = 0
- int **IOT_Change_Send_Functions.RMSSD_min** = 0
- int **IOT_Change_Send_Functions.RMSSD_max** = 0
- string **IOT_Change_Send_Functions.THINGSBOARD_HOST** = 'thingsboard.cloud'
- string **IOT_Change_Send_Functions.ACCESS_TOKEN** = 'REPLACE_With_YOUR_ACCESS_TOKEN'
- dictionary **IOT_Change_Send_Functions.temp_data** = {'temperature': 0}
- dictionary **IOT_Change_Send_Functions.BPM_data** = {'BPM': 0}
- dictionary **IOT_Change_Send_Functions.IBI_data** = {'IBI' : 0}
- dictionary **IOT_Change_Send_Functions.RMSSD_data** = {'RMSSD': 0}
- dictionary **IOT_Change_Send_Functions.image_data** = {'image':0}
- **IOT_Change_Send_Functions.next_reading** = time.time()
- **IOT_Change_Send_Functions.client** = mqtt.Client()

3.2.1 Detailed Description

In this sub-program, we developed certain functions to encrypt, change and send the collected data to the IoT platform.

Author

- Mr. Mohamed Ali Haoufa
- Mr. Mohamed Nacer Namane

Date

Jun 7, 2022

Copyright

3.2.2 Function Documentation

3.2.2.1 BPM_Sending_Condition()

```
def IOT_Change_Send_Functions.BPM_Sending_Condition (
              BPM,
              BPM_min,
              BPM_max,
              BPM_data )
```

Checks if the BPM value is within the specified range before sending.

This function checks whether the BPM value is within the specified range and sends the data to ThingsBoard if the condition is met.

Parameters

BPM	The BPM value to be checked.
BPM_min	The minimum allowed BPM value.
BPM_max	The maximum allowed BPM value.
BPM_data	The data to be sent.

Returns

True if the condition is met and data is sent, False otherwise.

3.2.2.2 cvt_2_base64()

```
def IOT_Change_Send_Functions.cvt_2_base64 (
              filename )
```

Converts an image file to a Base64-encoded string.

This function reads an image file and converts its content to a Base64-encoded string.

Parameters

filename	The name of the image file to be converted.

Returns

A Base64-encoded string representation of the image.

3.2.2.3 encrypt()

```
def IOT_Change_Send_Functions.encrypt (
              data,
              key,
              iv )
```

Encrypts the given data using AES encryption.

This function encrypts the given data using the AES encryption algorithm and returns the encrypted result.

Parameters

data	The data to be encrypted.
key	The encryption key.
iv	The initialization vector.

Returns

The encrypted data.

3.2.2.4 IBI_Sending_Condition()

```
def IOT_Change_Send_Functions.IBI_Sending_Condition (
              IBI,
              IBI_min,
              IBI_max,
              IBI_data )
```

Checks if the IBI value is within the specified range before sending.

This function checks whether the IBI value is within the specified range and sends the data to ThingsBoard if the condition is met.

Parameters

IBI	The IBI value to be checked.
IBI_min	The minimum allowed IBI value.
IBI_max	The maximum allowed IBI value.
IBI_data	The data to be sent.

Returns

True if the condition is met and data is sent, False otherwise.

3.2.2.5 image_generator()

```
def IOT_Change_Send_Functions.image_generator (
              data )
```

Generates and saves an image of the ECG signal.

This function generates an image of the ECG signal using the provided data and saves it as a PNG file. It also converts the image to a Base64-encoded string.

Parameters

data	The ECG signal data.

Returns

A Base64-encoded string representation of the generated image.

3.2.2.6 MAX_value()

```
def IOT_Change_Send_Functions.MAX_value (
              parameter,
              number )
```

Calculates the maximum value by adding a number to the parameter.

Parameters

parameter	The base value.
number	The number to be added.

Returns

The calculated maximum value.

3.2.2.7 MIN_value()

```
def IOT_Change_Send_Functions.MIN_value (
              parameter,
              number )
```

Calculates the minimum value by subtracting a number from the parameter.

Parameters

parameter	The base value.
number	The number to be subtracted.

Returns

The calculated minimum value.

3.2.2.8 RMSSD_Sending_Condition()

```
def IOT_Change_Send_Functions.RMSSD_Sending_Condition (
              RMSSD,
              RMSSD_min,
              RMSSD_max,
              RMSSD_data )
```

Checks if the RMSSD value is within the specified range before sending.

This function checks whether the RMSSD value is within the specified range and sends the data to ThingsBoard if the condition is met.

Parameters

RMSSD	The RMSSD value to be checked.
RMSSD_min	The minimum allowed RMSSD value.
RMSSD_max	The maximum allowed RMSSD value.
RMSSD_data	The data to be sent.

Returns

True if the condition is met and data is sent, False otherwise.

3.2.2.9 Temp_Sending_Condition()

```
def IOT_Change_Send_Functions.Temp_Sending_Condition (
              temp,
              temp_min,
              temp_max,
              temp_data )
```

Checks if the temperature value is within the specified range before sending.

This function checks whether the temperature value is within the specified range and sends the data to ThingsBoard if the condition is met.

Parameters

temp	The temperature value to be checked.
temp_min	The minimum allowed temperature.
temp_max	The maximum allowed temperature.
temp_data	The data to be sent.

Returns

True if the condition is met and data is sent, False otherwise.

3.3 src/main_ECG_Prog.py File Reference

It's the main program of our project, where we included all the sub-programs together to implement the following functionalities :

Variables

- main_ECG_Prog.arduino

  Serial port configuration ##.
- main_ECG_Prog.lectura = MCP3008(channel=2)

  Channel MCP3008 selection ##.
- dictionary **main_ECG_Prog.temp_data** = {'temperature': 0}
- dictionary **main_ECG_Prog.BPM_data** = {'BPM': 0}
- dictionary **main_ECG_Prog.IBI_data** = {'IBI' : 0}
- dictionary **main_ECG_Prog.RMSSD_data** = {'RMSSD': 0}
- dictionary **main_ECG_Prog.image_data** = {'image':0}
- int main_ECG_Prog.order = 5

  Define some values ##.
- int **main_ECG_Prog.fs_of_filer** = 5000
- int **main_ECG_Prog.temp_min** = 0
- int main_ECG_Prog.temp_max = 0

  Change the intervals of each parameter if it has a remarkable change #####.
- int **main_ECG_Prog.BPM_min** = 0
- int **main_ECG_Prog.BPM_max** = 0
- int **main_ECG_Prog.IBI_min** = 0
- int **main_ECG_Prog.IBI_max** = 0
- int **main_ECG_Prog.RMSSD_min** = 0
- int **main_ECG_Prog.RMSSD_max** = 0
- string **main_ECG_Prog.imagename** = 'ECG_signal.png'
- string **main_ECG_Prog.filename1** = 'data_csv_record.csv'
- string **main_ECG_Prog.key** = '03GosECsvhxGtbo='
- string **main_ECG_Prog.iv** = 'gaqIlAht+3nZWw=='.encode('utf-8')
- main_ECG_Prog.data = str(arduino.readline()).strip(",.babcdef'rxfn\ ")

  Start : Save 3600 values equivalent to 30 s of data ###.
- main_ECG_Prog.time1

  Split and Calculate fs ##.
- **main_ECG_Prog.fs**
- main_ECG_Prog.filtred_data = fl.final_filter(data, fs_of_filer, order)

Filter data ##.
- main_ECG_Prog.BPM = round(int(BPM), 2)

  Process the ECG data ###.
- **main_ECG_Prog.IBI** = round(int(IBI), 2)
- **main_ECG_Prog.RMSSD** = round(int(RMSSD), 2)
- tuple main_ECG_Prog.temp = (lectura.value ∗ 3.3)∗100 + 0.3

  Read temperature ###.
- **main_ECG_Prog.temp_encrypted** = sn.encrypt(temp,key,iv)
- **main_ECG_Prog.BPM_encrypted** = sn.encrypt(BPM,key,iv)
- **main_ECG_Prog.IBI_encrypted** = sn.encrypt(IBI,key,iv)
- **main_ECG_Prog.RMSSD_encrypted** = sn.encrypt(RMSSD,key,iv)
- main_ECG_Prog.temp_flag = sn.Temp_Sending_Condition(temp,temp_min,temp_max, sn.temp_data)

  Test change of data and send it to ThingsBoard IOT platform #####.
- **main_ECG_Prog.bpm_flag** = sn.BPM_Sending_Condition(BPM,BPM_min,BPM_max, sn.BPM_data)
- **main_ECG_Prog.ibi_flag** = sn.IBI_Sending_Condition(IBI,IBI_min,IBI_max, sn.IBI_data)
- **main_ECG_Prog.rmssd_flag** = sn.RMSSD_Sending_Condition(RMSSD,RMSSD_min,RMSSD_max, sn.←↩ RMSSD_data)
- **main_ECG_Prog.b64_string** = sn.image_generator(filtred_data)

3.3.1 Detailed Description

It's the main program of our project, where we included all the sub-programs together to implement the following functionalities :

- Read and collect the ECG Data from the Serial Port and save it in the CSV file.
- Spliting and calculating the frequency sampling of the Collected data.
- Filtering and processing the ECG data.
- The Encryption of data and then sending it to the IoT platform.
- Testing the data to see if there is a remarkable change.

Author

- Mr. Mohamed Ali Haoufa
- Mr. Mohamed Nacer Namane

Date

Jun 7, 2022

Copyright

Copyright (c) 2022. All rights reserved.

3.3.2 Variable Documentation

3.3.2.1 arduino

```
main_ECG_Prog.arduino
```

Initial value:
```
=  serial.Serial(
    port="/dev/ttyUSB0", baudrate=9600, bytesize=8, timeout=2, stopbits=serial.STOPBITS_ONE
)
```

Serial port configuration ##.

3.3.2.2 temp

```
main_ECG_Prog.temp = (lectura.value * 3.3)*100 + 0.3
```

Read temperature ###.

Rounding values ###.

3.4 src/Resample.py File Reference

In this sub-program, we processed the ECG signal data and extracted the essential vital parameters.

Functions

- def Resample.process_signal (data, fs)
  Process a signal using the heartpy library.

3.4.1 Detailed Description

In this sub-program, we processed the ECG signal data and extracted the essential vital parameters.

Author

- Mr. Mohamed Ali Haoufa
- Mr. Mohamed Nacer Namane

Date

Jun 7, 2022

Copyright

Copyright (c) 2022. All rights reserved.

3.4.2 Function Documentation

3.4.2.1 process_signal()

```
def Resample.process_signal (
              data,
              fs )
```

Process a signal using the heartpy library.

This function processes a given signal using the heartpy library and extracts the heart rate-related parameters.

Parameters

data	The input signal data.
fs	The sampling frequency of the input signal.

Returns

bpm The calculated beats per minute.

ibi The calculated interbeat intervals.

rmssd The calculated root mean square of successive differences.

3.5 src/SampleRate.py File Reference

In this sub-program, we split the collected data into two types and used them to calculate the frequency sampling of the ECG signal.

Functions

- def SampleRate.split_and_calcul_fs (filename)

  Read data from a CSV file and calculate the sampling frequency.

3.5.1 Detailed Description

In this sub-program, we split the collected data into two types and used them to calculate the frequency sampling of the ECG signal.

Author

- Mr. Mohamed Ali Haoufa
- Mr. Mohamed Nacer Namane

Date

Jun 7, 2022

Copyright

Copyright (c) 2022. All rights reserved.

3.5.2 Function Documentation

3.5.2.1 split_and_calcul_fs()

```
def SampleRate.split_and_calcul_fs (
              filename )
```

Read data from a CSV file and calculate the sampling frequency.

This function reads data from a CSV file, extracts the time and data columns, and calculates the sampling frequency (fs).

Parameters

filename	The name of the CSV file containing the data.

Returns

time The time values extracted from the CSV file.

data The data values extracted from the CSV file.

fs The calculated sampling frequency.

3.6 src/Save_Data_Functions.py File Reference

This sub-program includes the functions needed to save and remove/delete data (CSV, png).

Functions

- def Save_Data_Functions.write_to_csv (data, filename)
  Write data to a CSV file.
- def Save_Data_Functions.remove_csv (filename)
  Remove a CSV file if it exists.
- def Save_Data_Functions.delete_png (png_name)
  Delete a PNG image file.

3.6.1 Detailed Description

This sub-program includes the functions needed to save and remove/delete data (CSV, png).

Author

- Mr. Mohamed Ali Haoufa
- Mr. Mohamed Nacer Namane

Date

Jun 7, 2022

Copyright

Copyright (c) 2022. All rights reserved.

3.6.2 Function Documentation

3.6.2.1 delete_png()

```
def Save_Data_Functions.delete_png (
              png_name )
```

Delete a PNG image file.

This function removes the specified PNG image file from the file system.

Parameters

png_name	The name of the PNG image file to be deleted.

Returns

None

3.6.2.2 remove_csv()

```
def Save_Data_Functions.remove_csv (
              filename )
```

Remove a CSV file if it exists.

This function checks if the specified file exists and is a regular file. If it does, the file is deleted. If the file does not exist or is not a regular file, a corresponding message is printed.

Parameters

filename	The name of the CSV file to be removed.

Returns

message A message indicating whether the file was deleted or not.

3.6.2.3 write_to_csv()

```
def Save_Data_Functions.write_to_csv (
              data,
              filename )
```

Write data to a CSV file.

This function appends a timestamp and data to a CSV file. It uses the current time as a timestamp, and the provided data is appended to the file along with the timestamp.

Parameters

data	The data to be written to the CSV file.
filename	The name of the CSV file to write to.

Returns

write_to_log The result of the write operation (1 for success, 0 for failure).

Index