Exterior Door Security with Key-Words and Face Detection by Processing Signals in Real-Time

Ahmet Gizik
ag7739 – N13012289
Real-Time DSP Lab Project

December 22, 2021

Abstract

Today, smart home appliances are controlled using different user interfaces and based on various input devices. In this project, an exterior door security system is implemented by processing camera and microphone signals in real-time. Implemented program sequentially detects two different keywords. After detection of the keywords, it starts processing the camera signal to detect the user's face. When a human face is detected, the program takes a photo of the user and saves it with the precise information of the entrance time. Speech is a natural and easy way for communication between humans and machines. However, smart device manufacturers use a limited set of words to control them so that the system is more affordable and accessible for potential customers. If a user says different words then the keyword system detection does not happen so the program does not take any action as it should be. In addition to the implementation of the voice command detection, a pretrained face detection model is also added to the project to strengthen the demonstration of the project.

Libraries and Functions

The project is implemented in python language by various libraries and functions

SciPy

SciPy is a free and open-source Python library used for scientific computing and technical computing.

Opencv

OpenCV is a cross-platform library using which we can develop real-time computer vision applications. It mainly focuses on image processing, video capture and analysis including features like face detection and object detection.

Some others

```
cv2.cvtColor and cv2.COLOR_BGR2GRAY
```

These functions are used for converting video signals' RGB values to black and white values.

```
cv2.CascadeClassifier
```

This function handles the object detection using Haar feature-based cascade classifiers is an effective way. Its used to detect face and eyes with its default data.

```
cv2.rectangle
```

The function is used to frame the detected face.

```
cv2.imwrite
```

The function saves the captured face and after after tagging the time information by the help of $time.strftime$

Object Detection using Haar feature-based cascade Object Detection using Haar feature-based cascade classifiers is an effective object detection method proposed by Paul Viola and Michael Jones in their paper, "Rapid Object Detection using a Boosted Cascade of Simple Features" in 2001. It is a machine learning based approach where a cascade function is trained from a lot of positive and negative images. It is then used to detect objects in other images. Here we will work with face detection. Initially, the algorithm needs a lot of positive images (images of faces) and negative images (images without faces) to train the classifier. Then we need to extract features from it. For this, Haar features shown in the below image are used. They are just like our convolutional kernel. Each feature is a single value obtained by subtracting sum of pixels under the white rectangle from sum of pixels under the black rectangle.

Similarity measuring

Program measures the similarity between the recording and the microphone signal to detect the keywords.

Cross-correlation is a measure of similarity between two signals. It works by sliding one signal across another and finding the optimal match. This is also known as a sliding dot product or sliding inner-product and is closely related to convolution. As Wikipedia notes, cross-correlation is most often used to search a long signal for a potential shorter, known signal and is commonly used in pattern recognition, computer vision, and beyond. Cross-correlation between the normalized microphone input and recorded data values in time domain was the main application carried out this project.

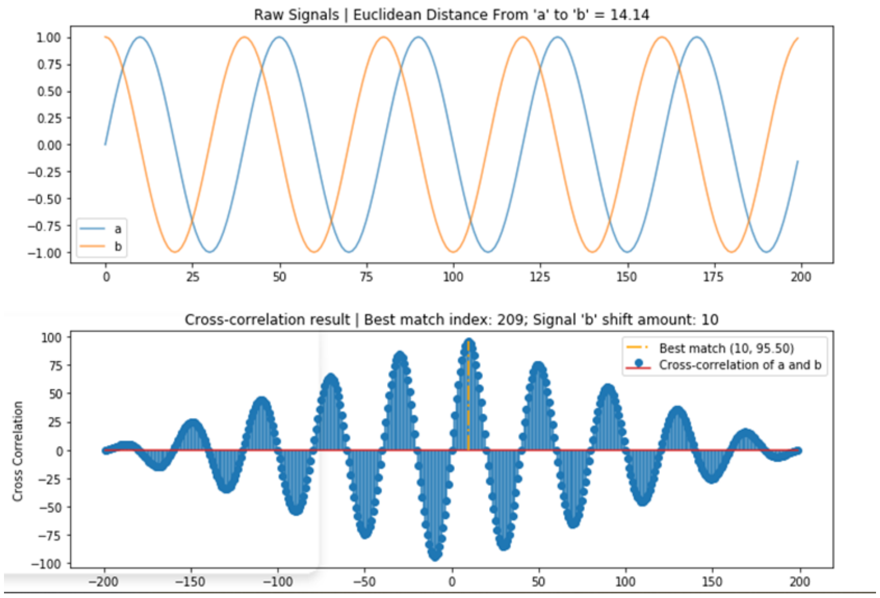


Figure 1: Best matching index by cross-correlation of sinusoidal signals as an example

Finding out the delay in number of samples was done by using maximum correlation point between the correlated signals.

There were other techniques I have tried and wasn't able to detect the keywords very well. Firstly, I have tried measuring the similarity with a phase independent matching by taking Fast Fourier Transform(FFT) of both microphone input and recordings to obtain a moving window of power spectrum values to have a linear time invariant metric that compares the shape and the size of the waveforms that is not sensitive to slow changes in phase between the waveforms that cause peaks to drift about. It did not work well.

My second trial again starting with taking the fast Fourier transform (FFT) of both signals to obtain a moving window of power spectrum values. Then calculating the mean square difference between the power levels for each frequency component in each window. Finally, taking the sum of these values for each window as the overall difference metric. This did not work as well.

Third trial was using dynamic time warping (DTW) algorithm, which is very much used in speech processing and other pattern matching applications. DTW is a popular method for measuring similarity between two signals in the time domain. It works by ”warping” the x-axis between two signals to find the closest y-value match.

Data Preparation

Data Preparation was an important step which includes windowing, normalizing and low pass filtering of the recorded signals and microphone input in this project.

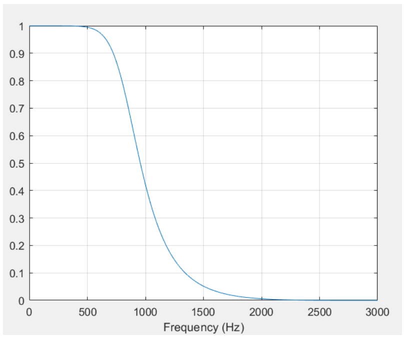


Figure 2: Frequency response of the applied LPF

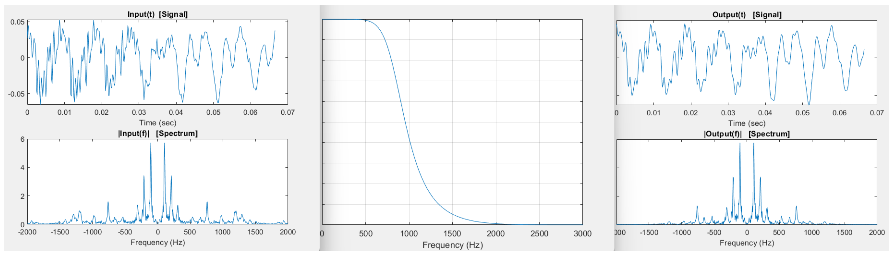


Figure 3: Application of LPF on one of the recorded signals

The cross-correlation function result depends on the recorded speech strength and the duration of recording. To perform cross-correlation independently of scale and duration the normalized cross-correlation is used.

Low pass is used to filter the noise so that the program can detect the keywords more accurately. Same low pass filter also applied on the microphone input block by block with saved states of the filter. Below, filter coefficients are listed and figure 2 shows the frequency response of the applied low pass filter.

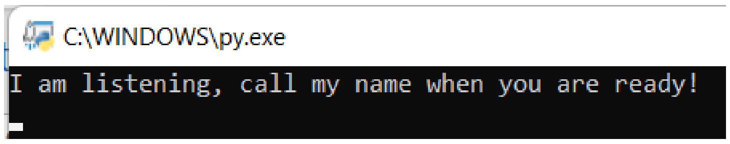
```
a =[1.0000,   -1.7027,    1.4164,   -0.5560,    0.0889]
b =[0.0154,    0.0616,   0.0925,    0.0616,    0.0154]
```

Program's Running Flow

The program processes the real-time audio for keyword detection from voice to decide whether the user is registered or not. Detection of the keywords are done by cross-correlation calculations between on going real-time input and pre-recorded user voice. Program also process the video in real-time, detects the user's face and saves the photo of the user with the date and time information after user's face is detected and framed.

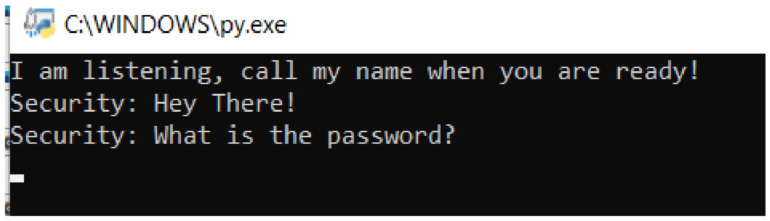
The main function only calls the scripts in the sequence of

```
1 - security.py
```



```
I am listening, call my name when you are ready!
```

Figure 4: Programs first output



```
I am listening, call my name when you are ready!
Security: Hey There!
Security: What is the password?
```

Figure 5: Programs second output

```
2 - banana.py
3 - facedetection.py
```

security.py python script is also a stand-alone project runs by clicking. When the door bell rings **main.py** calls the script called security, the program prompts the sentence below:

```
'I am listening, call my name when you are ready!'
```

Then assistant starts listening the guest or the resident, and asks for its name to start the human-machine communication. Here the assistans name is the first key-word program looks for and the initial password at the door. When the user call its name which is “Security” the program prints the sentences below and terminates itself.

```
'Security: Hey There!'
'Security: What is the password?'
```

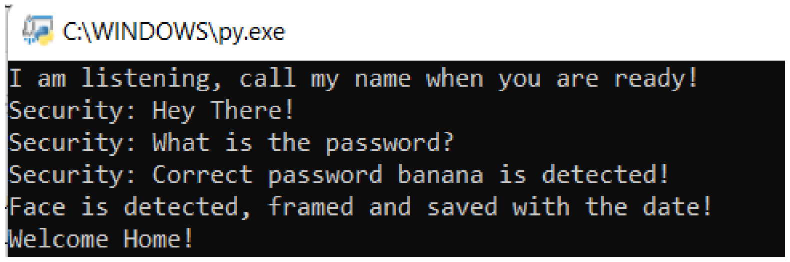
Then main.py calls the **banana.py** script which runs for detecting the second key-word. Program asks for the password and as soon as the user says “banana” script prints the sentences below:

```
'Security: Correct password banana is detected!'
```

After detecting the second key-word(password) banana.py script terminates itself. Lastly main.py calls the **facedetection.py** script. The last script looks for a human face. After the human face is detected and framed, it saves the image in the png file format with the entrance time information. At the end the program informs the user about taken face photograph and welcomes the user then terminates itself.

```
'Face is detected, framed and saved with the date!'
'Welcome Home!'
```

Figure 7 shows the taken photo of the user just before the program terminates itself.


C:\WINDOWS\py.exe

```
I am listening, call my name when you are ready!
Security: Hey There!
Security: What is the password?
Security: Correct password banana is detected!
Face is detected, framed and saved with the date!
Welcome Home!
```

Figure 6: Programs third output

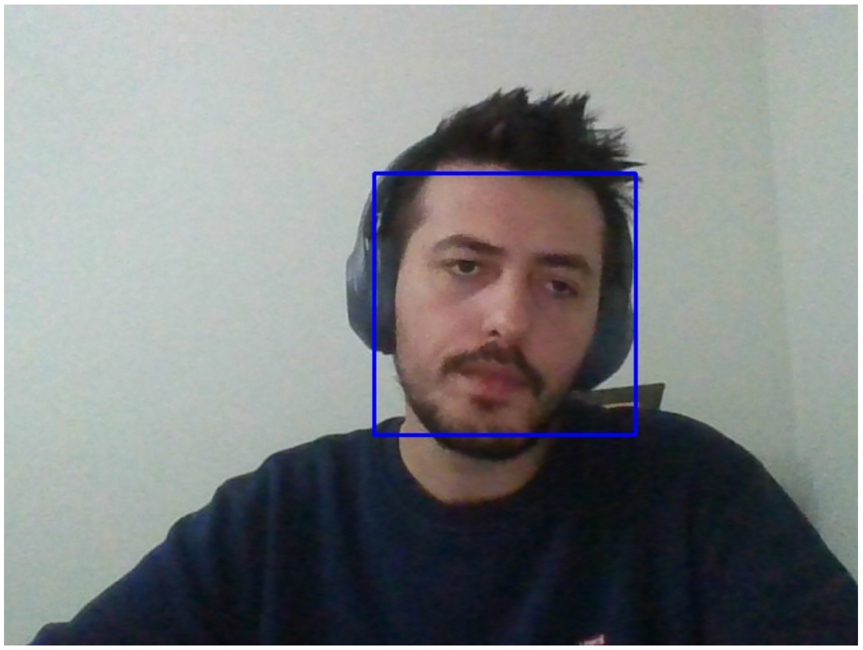
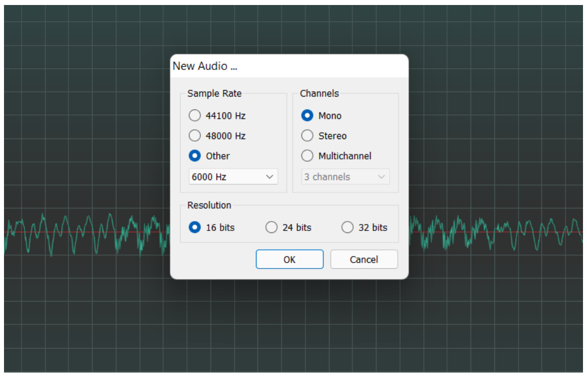


Figure 7: Users face framed and saved with the date


New Audio ...

Sample Rate: 44100 Hz, 48000 Hz, [✓ Other] 6000 Hz

Channels: [✓ Mono], Stereo, Multichannel (3 channels)

Resolution: [✓ 16 bits], 24 bits, 32 bits

Figure 8: Recording Values

Pre-recorded voice signals of the user are listed below. They are called by the program to compare them with the microphone inputs. Some of the files are parts of the recorded words which will be explained soon.

```
Recordings of the word "banana"
-b1.wav , b2.wav, b3.wav, b4.wav, b5.wav, b6.wav

Recordings of the word "security"
-1.wav , 2.wav, 3.wav

Recording parts of the word "security"
-p1.wav , p2.wav, p3.wav
-1_2.wav, 2_3.wav
```

As it is explained before, main.py just calls the other scripts. It stars with calling the security.py script but let's start with explaining banana.py which is less advanced then security.py.

Figure 9 shows the voice signal of a human saying the word "banana". The signal below is recorded with 6000 Hz sampling frequency, 16 bits resolution and with a mono channel. Here notice that low sampling rate, low resolution and mono channel instead of a higher sampling rate, a higher resolution and stereo recording type is used during the recording. Therefore the hardware and product will be cheaper to manufacture since it requires just a basic microphone, sampler and decoder.

The word banana is easy to detect because it is continuous one part signal which makes measuring the similarity with the input signal easy.

How the code works?

One block size is half of a prerecorded signal size. There are also three buffer to keep the past three blocks so that the program is able to process a length of twice recorded signal length samples. Here the recorded signal length is approximately 0.5

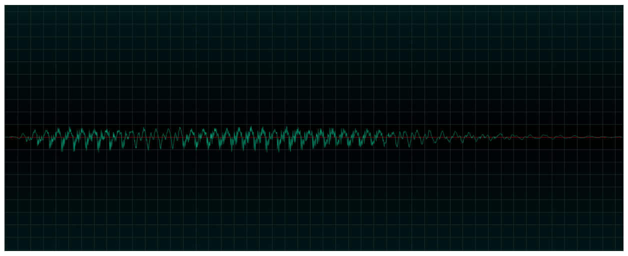


Figure 9: Recorded voice signal of a user for the word ”banana”

second(windowed pre-recorded signal) so the block duration is approximately 0.25 sec. which provides a very short delay.

```
pre_recorded1 = signal.lfilter(bfilt, afilt, pre_recorded1)
BLOCKLEN = int(signal_length1/2)
```

Cross-correlation of the recorded signal and real-time microphone signal in the code as below:

```
input_bytes = stream.read(BLOCKLEN) # BLOCKLEN = number of frames read

input_tuple = struct.unpack('h' * BLOCKLEN, input_bytes)

[last_taken_samples1, states] = signal.lfilter(bfilt, afilt,\
input_tuple, zi = states)

last_taken_samples1 = np.array(last_taken_samples1,\
 dtype='float')


last_taken_samples1 = normalize(last_taken_samples1)



last_taken_samples4 = np.concatenate((last_taken_samples3,last_taken_samples2,\
last_taken_samples1), axis=0) #one dimensional array


cr1 = signal.correlate(last_taken_samples4, pre_recorded1, 'full');
```

Then checking the maximum correlation index to detect the shift between them. Programs tries to catch the b1 index in the middle of my total buffer so that it can measure the maximum correlation otherwise it can catch the half of the signal in the most current block or in the latest part of the buffer.

```
b1 = np.argmax(np.abs(cr1));
```

```
if b1 > BLOCKLEN and b1 <  BLOCKLEN*3:

    cond1 = 1;
    arr1 = last_taken_samples4[b1 - BLOCKLEN: b1 + BLOCKLEN]
    r1 = np.correlate(arr1, pre_recorded1, 'full');
    result1 = np.max(np.abs(r1))
```

Measuring the cross-correlation between signals and record the score:

```
total =cond1*result1 + cond2*result2 + cond3*result3+\
cond4*result4 + cond5*result5 + cond6*result6

tot_div = cond1 + cond2 + cond3 + cond4 + cond5 + cond6

max_corr = np.max([cond1*result1 , cond2*result2, cond3*result3\
,cond4*result4 , cond5*result5 , cond6*result6])


if max_corr > 40:
    mean_corr = total/tot_div
    if mean_corr > 80:
        print("Security: Correct password banana is detected!")
        print("Face is detected, framed and saved with the date!")
        print("Welcome Home!")
        break

last_taken_samples3 = last_taken_samples2
last_taken_samples2 = last_taken_samples1
```

Lastly, checking the correlation score of signal and 6 different recordings to decide whether the keyword is detected or not, then updates the buffer and wait for the upcoming block.

Security is different

In figure 10, the voice recording is shown which is the signal in time domain while the user saying ''security" which is approximately 0.75 sec.

```
In code : pre_recorded = '1.wav'
```

As you can see in figure 10, unlike the word banana, security has 3 parts in it which makes it much comlicated. If we try to catch all the word at once, we could correlate with any other continious signal more than the actual key word. Therefore it requires analyzing the at 3 parts and sequentially since this is a time series data.

```
Part 1
In code p1file = 'p1.wav'
```

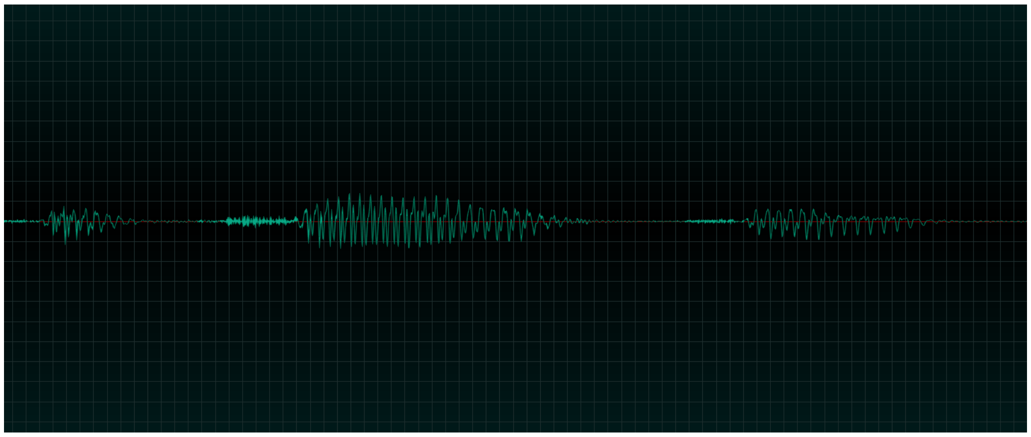


Figure 10: Recording of Security

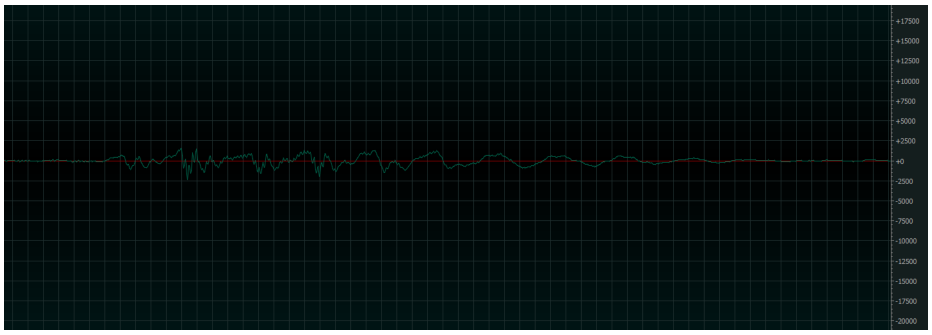


Figure 11: Recording of First Part of Security

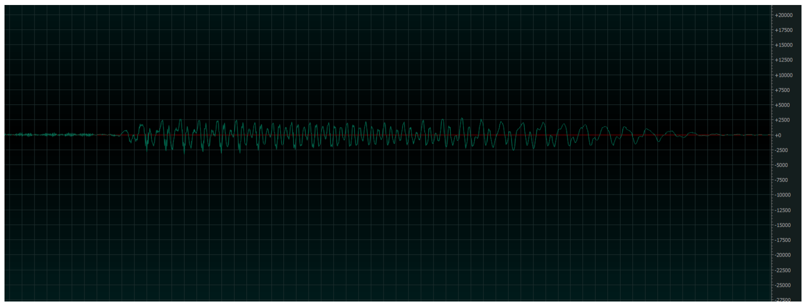


Figure 12: Recording of Second Part of Security

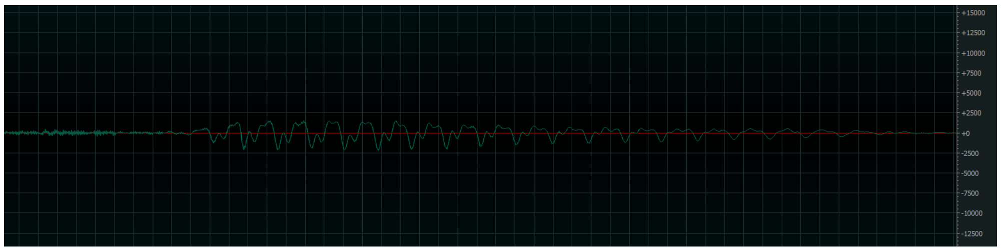


Figure 13: Recording of Third Part of Security

```
Part 2
In code p2file = 'p2.wav'

Part 3

In code p3file = 'p3.wav'

Part 1 and 2
In code p1_2 = wavfile.read('1_2.wav');

In code
```

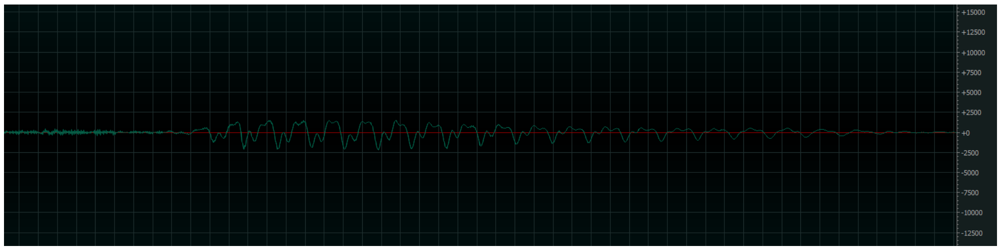


Figure 14: Recording of First-Second of Security

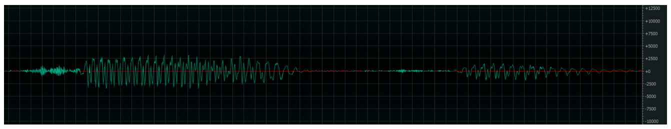


Figure 15: Recording of Second-Third Part of Security

```
p2_3 = wavfile.read('2_3.wav')
```

The signal and parts are shown in figures are raw recordings, the low pass filter is applied on them and normalized before measuring cross correlation with the input signal. Same filter is also applied to microphone signal:

```
[input_tuple, states] = signal.lfilter(bfilt, afilt, input_tuple, zi = states)
```

Below we will see that similarity measurements and maximum correlation indexes are checked. Here program calculates the maximum correlation index for the first part. It only looks after a quarter block length which is a 1/8 signal length so that it does not detect the key-word multiple times, does less computations and consumes less power.

```
r55 = signal.correlate(last_taken_samples4[half_idx:], p1, 'full');
b5 = np.argmax(np.abs(r55));
a = r55[b5]
b5 = b5+half_idx;
```

Here, program checks the index's position which gave the best correlation. If the index is smaller than a block size which makes sure that it is between 0.25*blocklen(0.125*signal length) and 2*blocklen(1.125*signal length) range since halfidx(quarter block length) is added before this check. Note that here program is looking for the first part of the signal which is approximately quarter length of the whole signal since there values are approximately zero between first and second part of the signal. The range is only 1 signal length in 2 blocks which makes sure that the program will catch the part.

Case 1: Comes at the beginning of the block, it will be detected in the third block's head. Case 2: Comes at the end of the block, it will be detected in the second block's head.

Other two parts' cases are similar below and same intuition is applied while coding. If b5 ¡ 2.5 blocklen, second part definitely will be detected before 3.5 block otherwise we will catch the keyword when the next block of samples is taken.

```
if b5 < BLOCKLEN*2 + half_idx:

    p1_score = np.max(signal.correlate(last_taken_samples4[b5 -\
    half_len_p1 : b5 + half_len_p1], p1, 'full'))


    p12_score = np.max(signal.correlate(last_taken_samples4[b5\
    -half_len_p1 : b5 + half_len_p1_2 + half_len_p1], p1_2,
    'full'))
    #print('part 1_2 score:',p12_score)

    p2_loc = b5 + half_len_p1



    r66 = signal.correlate(last_taken_samples4[p2_loc:(2*BLOCKLEN\
  + idx)], p2, 'full');

    b6 = np.argmax(np.abs(r66));
    b = r66[b6]
    b6 = p2_loc + b6;

    if b6 <  3*BLOCKLEN + idx:

        p2_score = np.max(signal.correlate(last_taken_samples4[b6 -\
half_len_p2 : b6 + half_len_p2], p2, 'full'))

        p23_score = np.max(signal.correlate(last_taken_samples4[b6 -\
half_len_p2 : b6 + half_len_p2_3 + half_len_p2], p2_3, 'full'))

        #print('part 2_3 score:',p23_score)

        p3_loc = b6 + 200
        r77 = signal.correlate(last_taken_samples4[p3_loc:], \
        p3, 'full');


        b7 = np.argmax(np.abs(r77));
        c = r77[b7]
        b7 = p3_loc + b7;
```

When the similarity measure is high enough for the first part, program checks and detects the sample delay between correlated numpy arrays at the same time it measures the correlation value and checks the tresholds.

```
if 4*BLOCKLEN - half_len_p3 >  b7:

    p3_score = np.max(signal.correlate(last_taken_samples4\
    [b7 - half_len_p3 : b7 + half_len_p3], p3, 'full'))
    #print('--------------case1-----------------------part 3 score:',p3_score)

    if p23_score > p12_score:
        arr2 = last_taken_samples4[b6 - idx: b6 + idx]
        r61 = np.correlate(arr2, pre_recorded, 'full');
        r62 = np.correlate(arr2, pre_recorded2, 'full');
        r63 = np.correlate(arr2, pre_recorded3, 'full');

    else:
        arr2 = last_taken_samples4[b5 - half_len_p1 : b5 +\
        BLOCKLEN + half_len_p1]
        r61 = np.correlate(arr2, pre_recorded, 'full');
        r62 = np.correlate(arr2, pre_recorded2, 'full');
        r63 = np.correlate(arr2, pre_recorded3, 'full');


    r6 = np.concatenate((r61,r62, r63), axis=0) #1D array

    a = np.max(r61)
    b = np.max(r62)
    c = np.max(r63)

    arr = [a, b, c]
    if(np.mean(arr)>75 and np.min(arr) > 65):
        print("Security: Hey There!")
        print("Security: What is the password?")
        break

            else:

    #print('-----case2---do-not--print--but--save---the---log-------')
    if p23_score > p12_score:
        arr2 = last_taken_samples4[b6 - idx: b6 + idx]
        r61 = np.correlate(arr2, pre_recorded, 'full');
        r62 = np.correlate(arr2, pre_recorded2, 'full');
        r63 = np.correlate(arr2, pre_recorded3, 'full');
        log.append(p23_score)
        log_val.append(n)

    else:
        arr2 = last_taken_samples4[b5 - half_len_p1 : b5 +\
        BLOCKLEN + half_len_p1]
        r61 = np.correlate(arr2, pre_recorded, 'full');
        r62 = np.correlate(arr2, pre_recorded2, 'full');
        r63 = np.correlate(arr2, pre_recorded3, 'full');
        log.append(p12_score)
        log_val.append(n)


    r6 = np.concatenate((r61,r62, r63), axis=0) #1D array

a = np.max(r61)
```

```
b = np.max(r62)
c = np.max(r63)
d = np.max(np.correlate(last_taken_samples4, pre_recorded,\
'full'));

a1 = np.min(r61)
b1 = np.min(r62)
c1 = np.min(r63)

arr = [a, b, c]

last_taken_samples5 = last_taken_samples3
last_taken_samples3 = last_taken_samples2
last_taken_samples2 = last_taken_samples1
```