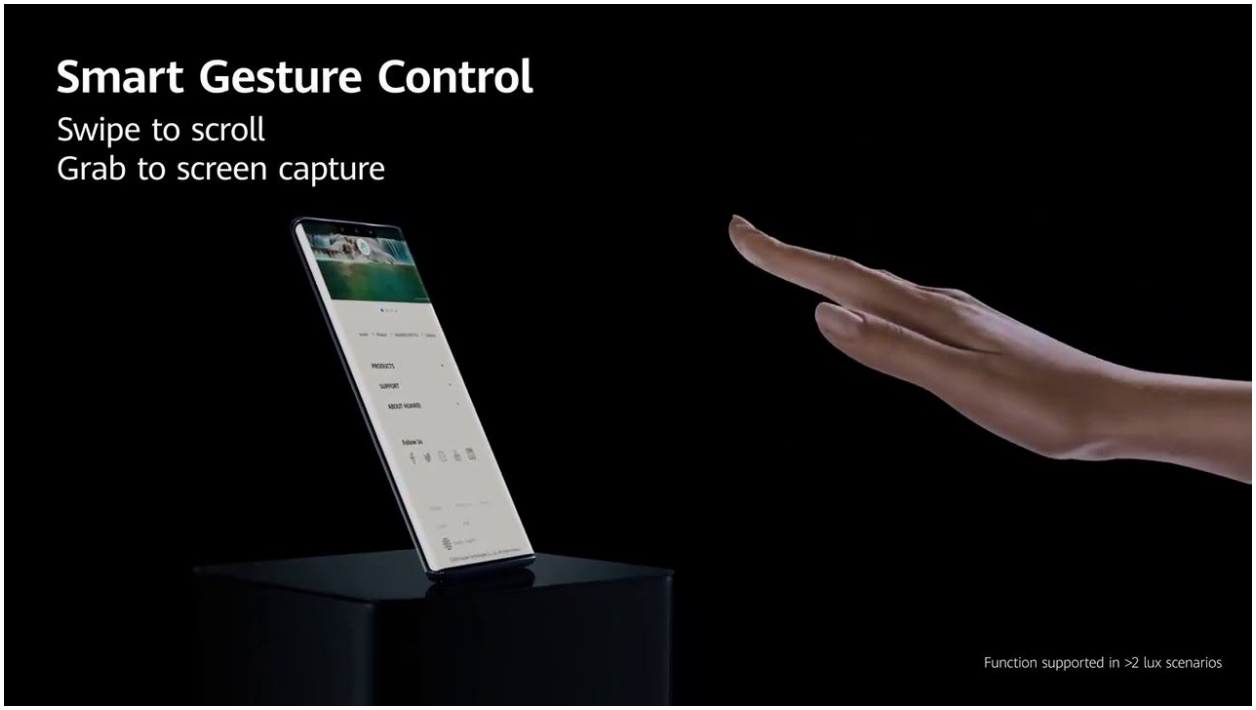
Multimodal Interaction: Smart Device Gesture Communication



Course: Multimodal Interaction

Master in Computer Science “La Sapienza Università di Roma”

Academic Year: 2019-2020

Professor: Maria de Marsico

Made by: Anxhelo Diko

Content

1. Introduction

In today's fancy world we interact a lot with smart devices, like alexa or google home, etc. We communicate with them using our voice and our hearing system, it is so natural but for a lot of people (deaf and mute) it is so unnatural. The proposed project tries to create a communication bridge between smart devices that have voice as their primary communication channel and mute and/or deaf people. The system is designed to take input hand gestures, translates them into a command (mute people benefit this part) for smart devices and then it waits for their voice feedback and through speech recognition recognizes the feedback and shows it as text on the GUI (deaf people benefit this part). On Figure 1 is shown a schema of the application workflow.

It has two main modules:

1) Video module: Capture the frame that contains the gesture from the camera.
2) Audio module: Makes possible the text-to-speech and speech-to-text part.

All modules were realized with python 3.6, no other language was used. The main libraries used are OpenCV 4.3.0, speech_recognition (GOOGLE API), pytorch (for building the classifier), Sound eXchange (sox), pyaudio, imutils, google_speech (uses sox as backend to produce sound from text). Because this is a prototype the number of gestures is very small also because the dataset was self made. The drawback that it has with so few gestures is that in this case on this prototype we can put as many commands.

The goal for this project is to make persons with limited capabilities (mute and deaf) interact with smart devices that use speech as their interaction language. Goal is what we want to achieve, is very high level so to better design it we need to split it into tasks and complete them one by one to later join them together. The main tasks identified for this work that serve our goal are as follow:

- User Gesture recognition (recognize gestures)
- Text to voice command conversion (text to speech conversion)
- Speech recognition (speech to text conversion)

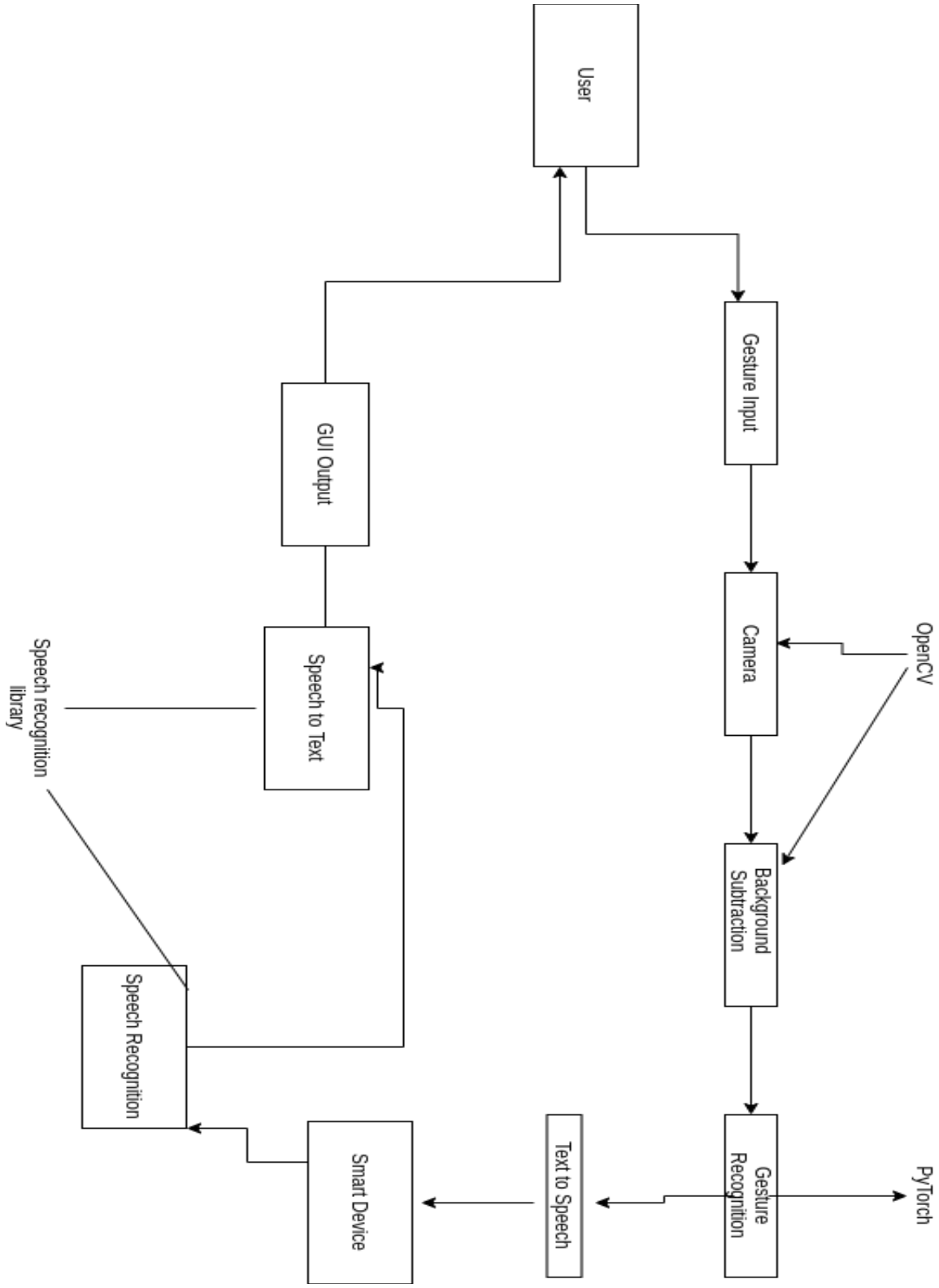


Figure 1 : Workflow of the application

2. Per task use cases

In this section will be shown only the schemas of use cases, the details will be explained on the implementation part.

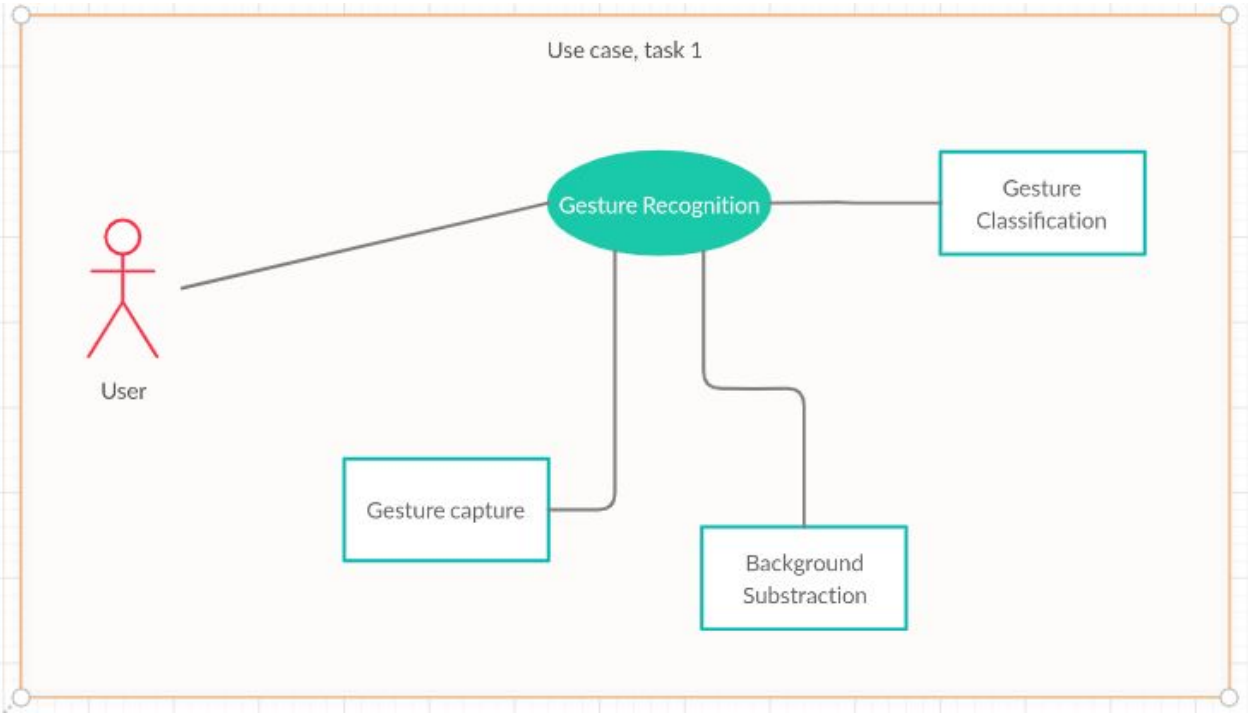


Figure 2: Use Case 1, task 1-User gesture recognition

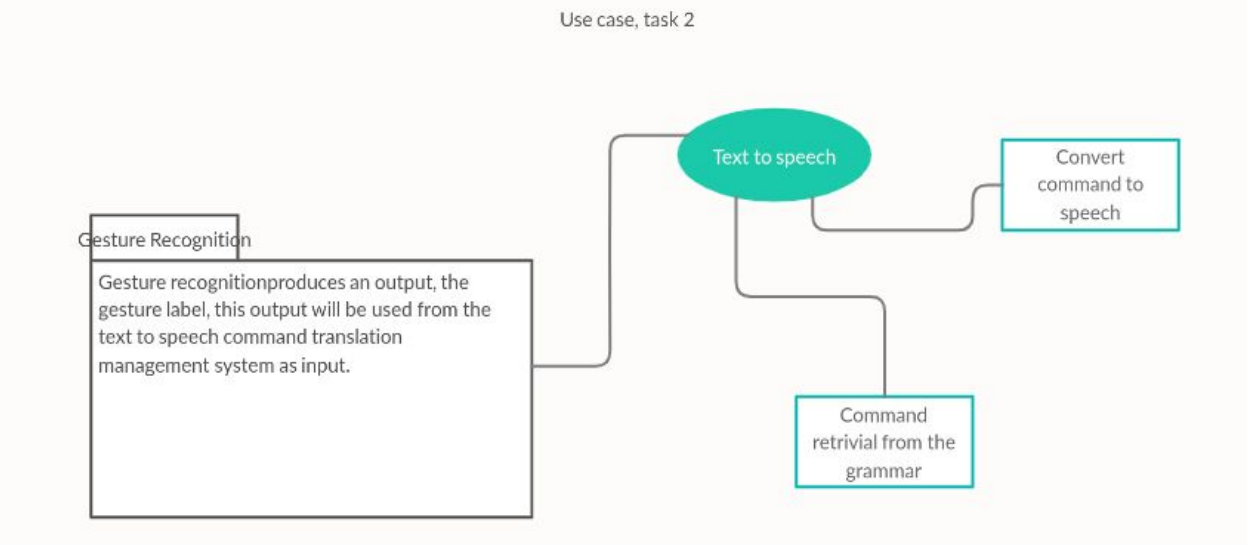


Figure 3: Use Case 2, task 2-Text to speech command conversion

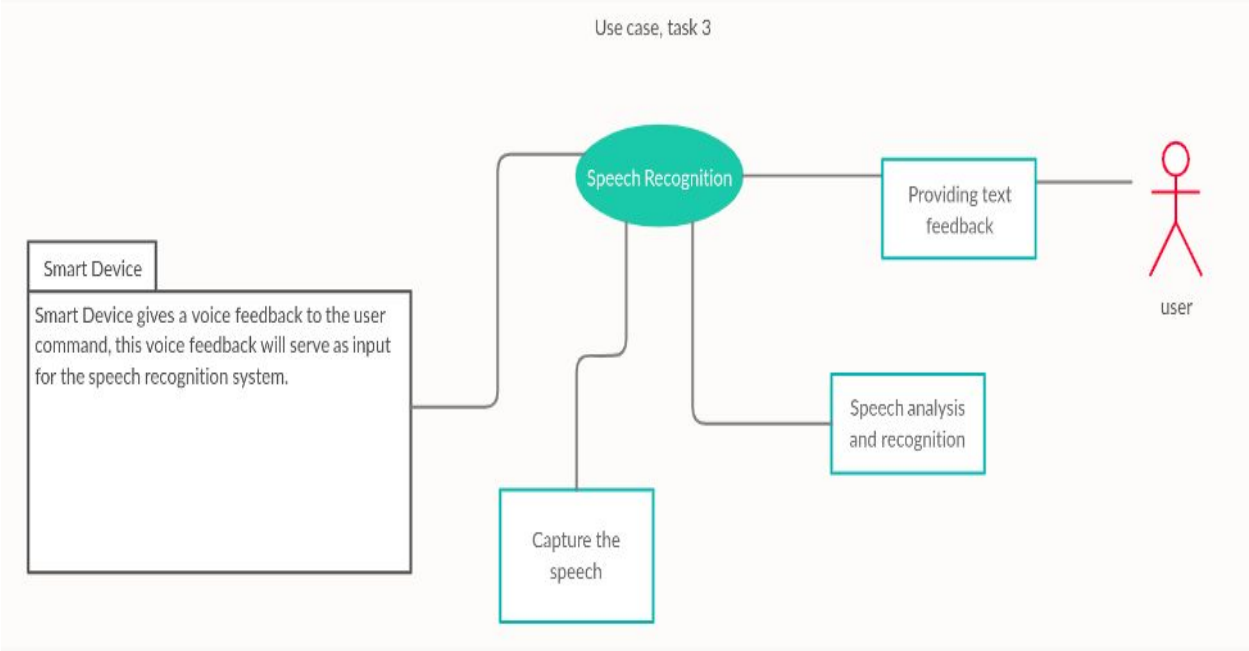


Figure 4: Use case 3, task 3-speech recognition

3. Task 1

The first use case is about video modules and consist in user input capture and recognition. So the input channel is the camera and the user has to show the gesture to the camera. Now, before all, as soon as the application starts the camera captures the background and after that it is constantly recording 5 frames per second to capture the gestures in real time. On the UI there is a green square that indicates where the user should show the gesture to the system. As soon as the system captures the gesture it passes it to the function that does background subtraction and then to the classifier for gesture classification. The gesture classification computes the confidence that the captured gesture is recognized, if the confidence is bigger than a settled threshold then the system proceeds with the command. Now this is the automatic way of giving commands, but there are cases where the camera captures a lot of noise and the gesture confidence finds difficulties to pass the threshold, for this case the users have a possibility to press a button and send the command related to the recognized gesture no matter the confidence value.

3.1 Technical details for use case 1

Video module management starts with the camera. We manage the camera through OpenCV 4.2.0.1 through python API. The system directly takes control of the main camera device or the default.

```
camera = cv2.VideoCapture(0)
(grabbed, frame) = camera.read()
```

So now we capture frames in real time. To begin with as we stated before we capture the background.

```
if not (os.path.exists(os.path.join(os.getcwd(),'background.jpg'))):
        cv2.imwrite('./background.jpg', img=roi)
        cv2.waitKey(100)
        cv2.destroyAllWindows()
```

From this point we are ready to proceed normally. As soon as the system captures a gesture it computes the background subtraction with the following piece of code.

```
def getBackgroundSubstraction(background, frame):

    """"
    input: background image and the actual image
    output: segmented image with subtracted background
    """

    g_background = cv2.cvtColor(background, cv2.COLOR_BGR2GRAY)
    g_foreground = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)

    dif = cv2.absdiff(g_background, g_foreground)
    _,result = cv2.threshold(dif, 25, 255, cv2.THRESH_BINARY)

    return result
```

The result is as follows:

Figure 4: Background subtraction

Now the extracted gesture will serve as input to a classifier. The classifier is a simple CNN built with pyTorch framework. It takes grayscale images that result from the gesture extraction and computes to which class it is most likely to belong to.

```
with torch.no_grad():
        prediction = model(tensor_image)
        pred = prediction.max(1, keepdim=True)[1]
        pred = np.array(pred)
        prediction = np.array(prediction)
```

In the above piece of code, the tensor image is the extracted gesture converter into a torch tensor, the model() is a CNN object that takes as input the torch tensor and returns an array with the likelihood that the feeded image has to belong to each class. Then prediction.max selects the label with the highest likelihood. After the label is assigned the confidence of the label or the likelihood is compared to a settled threshold for automatic interaction without any key pressed with the following piece of code.

```
if (labels[int(pred[0][0])] != 'nothing'):
            if(labels[int(pred[0][0])] == 'L'):
                if prediction[0][int(pred[0][0])] >12.0:
```

If the threshold is reached then the command is sent.

```
message ="Command:" +commands[labels[int(pred[0][0])]]
speech.text_to_speech(commands[labels[int(pred[0][0])]])
```

Now there are cases when the camera captures a lot of noise and the threshold is too difficult to be reached (as we said before), to make possible that the user can still interact with smart devices we have an option that forces the command, it should press the space key from the keyboard.

```
if keypress == 32: #if space is pressed
        print("Command:",commands[labels[int(pred[0][0])]])
        speech.text_to_spech(commands[labels[int(pred[0][0])]])
```

3.2 Important decisions when dealing with video module:

1) Inserting a square in the gui where the user should show the gesture. This decision was set by keeping in mind the computer vision limitations and the user experience point of view. The computer vision limitations in this case vary from the quality of camera. By capturing the entire frame and then cropping the hand we would provide a better user experience of course, but in this kind of system and application the accuracy is really important. With the entire frame we would add another layer of uncertainty to the system when knowing that it is not that easy to extract the hand from the other parts of the body on different occasions based on the hand position with respect to the other parts.
2) Adding a button to force the command. This was to avoid a long time of waiting for noisy frames to reach the threshold. A bit of more effort from the user is required to improve the overall user experience, also is very important to notice that provides the possibility to use the application even in low quality cameras.
3) How the threshold was chosen. We chose the threshold high enough to avoid noisy frames, give false positive gesture classification and send commands without the user's will.

4. Task 2

Use case 2 shows how the text to speech part is handled. This task is highly related with the previous task , task 1 as it takes as input the output of the video module. The command coming from the first task is in text format and in order to transmit it to the smart device it is necessary to convert it into audio format. For implementing this part we have used google api called google_speech which uses SoX library as a backend. SoX stands for Sound eXchange which is a cross platform library for audio editing.

4.1 Technical details for task 2

SoX can take a text and a language as input and produce audio or sound format of the provided text in the provided language, the language should be the same as the text, it does not do translation. As we said earlier we call SoX through google with the following piece of code:

```
def text_to_spech(text=None):
    """
    Text-To-Speech from its name takes input a string or a text and
transforms it into audio
    .Speech function is google function that takes as input the text and
reads it, can work on and offline
    """
    sox_effect1 = ("delay", "0.5")
    text_to_spech = google_speech.Speech(text, 'en')
    text_to_spech.play(sox_effect1)

    return
```

Let's start with the parameter named text. This parameter will be the output retrieved from the first task. So it will be a string or text format. Then we have sox_effect1 that defines the delay of the voice before being produced, by default it was 1 second but the delay was a bit noticeable. Another important factor to take in consideration in the case of text to speech was the speed of sound reproduction, the default speed was good in our case but it should chosen carefully because a lot of smart devices takes the input in chunks where a chunk represent a sentence and a low speed reproduction of the sound may cause that the distance between two words is recognized as a full stop and the smart devices that has a limit of chunks per command will perform very poor. As we can see the Speech() function takes two parameters text and language as we said before, the default language is english (british english). In the end when we do play, after the previous instruction has created the audio, we also pass the sox_effect to customize the sound. In the end we call this functionality the following way:

```
speech.text_to_spech(commands[labels[int(pred[0][0])]])
```

And obviously the command represents the output from the first task.

4.2 Important decisions when dealing with audio module for sound producing:

1) Delay factor. More it was a user point of view, to make the sound start faster, it does not seem much of a delay with the default value of 1 but hearing it the difference is noticeable.
2) The speed of reproduction. It was explained above.

5. Task 3

The third task is speech recognition, and the final task to complete the entire functionality of the software and reach the goal. Let's start with the input. The input provided to this task that deals with audio modality also, like the previous task but with different goals, is the answer that the smart device gives with a spoken language to the given command from the first and the second task. The first subtask is to capture the speech. This part is very tricky because an acoustic environment is never stable, without noise. Also device incorporated microphone are exposed to device internal noise, for example the cooling fan inside the pc, this makes it difficult to extract the right information from the input speech. So it is crucial to calibrate the recorder before recognizing the speech. The second subtask is speech recognition, this will be handled by google api, we just need to provide the parameters. It is an AI base recognizer that achieves state of the art accuracy and stands among top 3 systems together with IBM and AMAZON.

5.1 Technical details for task 3

The way we dealt with it is straightforward, before the system starts recording the speech, it accumulates some statistics about the environment noise to set a proper noise threshold for separating the environment random noise from the actual speech. We have used google speech_recognition API to handle this part with the following line of code:

```
r.adjust_for_ambient_noise(source)
```

In the above line of code source is the source microphone, the one that will record the speech, the line of code, obviously does the noise adjustment. For better recognition we decided to call this line of code before every speech recognition process because the noise is not something static but changes very frequently. Something to be mentioned here is timing. In this part of processing it is very important. It is important that the

system finishes the noise adjustment before the smart device starts to produce the voice feedback. The entire piece of code that handles the speech recognition and represents the 3rd use case is the following:

```
def speec_to_text():
    """

    This function does not take any input and does speech recognition
itself.
    sr.Recognizer() is the function of speech_recogniton that does the
voice to text convertion
    sr.Microphone() is the function that uses the device microphone
    r.adjust_for_ambient_noise() adjustes the lower threshold of noise
based on the environment noise
    r.listen() is the function that records a phrase and can wait up to 10
seconds if no phrase is said
    recognize_google() is the function that decides which recogniser we
going to use and it works online


    """

    r = sr.Recognizer()

    with sr.Microphone() as source:
        print("enters recording")
        r.adjust_for_ambient_noise(source)
        print("waiting for input")
        audio = r.listen(source, timeout=5)

        print("recognizing the input")

        # jsonCred = json.loads(GOOGLE_APPLICATION_CREDENTIALS)
    try:
        text = r.recognize_google(audio)
        print("You said: " + text)    # recognize speech using Google
Speech Recognition
```

```
    except LookupError:                                 # speech is
unintelligible

        print("Could not understand audio")

    return text
```

From the code it can be seen that we use google speech recognition, not the cloud based one but the google search engine speech recognizer. There are two possibilities using this recognizer, one is the free of charge modality where u have 60 recognitions per day, and another one is by attaching the software with a google cloud project and generate a credential key that will allow you to use the api or the service unlimited but payable. A drawback of this modality is that it always works online, the only offline method or tool for speech recognition is sphinx but sphinx works very “bad” compared to the other api options. Sphinx is offline as we said but only accepts good british english accents compared to the multilingual support that the other apis offer. Is important to mention that google speech_recognition library works on top of pyAudio for catching the speech. Now after the speech recognition part is done, an important thing is to show the feedback of the smart device on text format in the UI. We have used a library called tkinter to do so.

5.1 Important decisions when dealing with audio module for speech recognition:

1) Time out value. We set the timeout to 5 seconds, it is arbitrary, but after 5 seconds if the smart device does not send a feedback means that there is a problem, usually it would be a connection problem otherwise it would be strange to have a late reply.
2) Number of chunks (sentences) it can recognize. Our system can recognize up to two sentences, it can be changed very easily but after a lot of proves we found this sufficient, because in more than 90% of cases in our tests, the smart device responded with at most 2 sentences plus the speed of speech production of the smart devices very often makes all the sentences being recognized as one chunk but then with the help of NLP (google api uses it in the backend) it is splitted in the appropriate chunks or sentences.
3) The important decisions made on the second task were directly related to this task.

6. Problems during implementation

The problems encountered in this work were too many. Most of them were due to computer vision limitations (even audio preprocessing limitation but i used Google API to deal with them as i have lack of experience working with that kind of data) but this problems were mentioned in the above sections, one problem that was not mentioned was the dataset for training the classifier. I initially started to build a model that worked with rgb images directly using the asl dataset (american sign language) but during the working period my camera showed some problems and was not consistent in the quality of frames and noise captured so i had to change strategy. After this I decided to create my own dataset inspired by a github repository https://github.com/athena15/project_kojak. I used this guy's dataset together with pictures of mine to create a dataset with around 4-5 hundred images per sign and trained a classifier. So the dataset was partially created and partially borrowed from the internet.

7. Future improvements

This might have been the beginning of an interesting topic of research as it is very challenging and beautiful at the same time. My future scope will be to integrate sign language translators and not just use gestures to give the opportunity of a full expression to deaf and mute people to interact as normal and natural as possible with smart devices.

8. Conclusions

Deaf and mute oriented products are very few because of technology limitations that we have in the present. Integrating different technologies and modules like vision and audio is easy to imagine but very hard to do. This was a simple approach with modest implementation where an idea about how to improve deaf and mute people interaction with smart devices using computer vision, speech recognition and speech was introduced. Also to mention is that even with simple ways the problems that arise are too many and of course there is a huge room for improvement.