CHAPTER - 4

SYSTEM IMPLEMENTATION

4.1 INTRODUCTION

The disadvantages of the existing system in Air Traffic Services, had been discussed earlier. One disadvantage that we focus is “human stress”. In order to overcome this particular disadvantage, we are proposing a new solution to it in a creative way. The solution is framed in the domain of Data Mining and Machine Learning. The proposed system is mainly a solution to eliminate the occurrence of human errors due to physical and mental stress. The system is implemented by modularizing it into 4 parts depending on its working. The factors considered for making these 4 modules were the entities involved in a conversation between a pilot and a controller. The entities are: Pilot, who requests controller for landing/takeoff – approach/clearance, our controller bot, and a human controller who supervises the working of bot. All the 4 modules comes under 3 sub-divisions: Data Collection, Data Pre-processing and Data Analysis. Module 1 is also given a name “Pilot request speech to text” where the pilot entity and controller bot are involved. Module 2 is named “Text Summarization” which is a vital part of Controller bot and accomplished by implementing LSTM. Module 3 is named “Response Generation” which also involves stop words removal in it. Module 4 is the last module which is “Text to Pilot response” which also includes a sub-module called “Flight information inclusion”, where controller bot sends response to pilot’s request. In existing system he is responsible for safety maneuvering of aircraft. The standard method of communication between an air traffic controller and a pilot is voice radio, using either Very High Frequency (VHF) bands for line-of-sight communication or High Frequency (HF) bands for long-distance communication (such as that provided by Shanwick Oceanic Control). Fig 4.1 represents voice radio communication.

Fig 4.1 Voice radio Communication

Courtesy: Arabian Airspace

The busy controller in main control tower receives the request from the pilot and transfers the response. The main hardware components that we are going to use are shown in fig 4.2. A good performing Personal computer with internet connectivity and earphones for passing the input.

4.2 SYSTEM MODULES

The proposed system is segregated into 4 main modules, which are

- Speech request to text
- Text Summarization
- Response Generation
- Text response to Speech

All the above modules make up our system modules. These modules are individually made up for working. Each and every module possess an individuality in working and contributes equally for the overall working of our project. The required hardware components for our work is, a good featured personal computer and headphones/earphones for passing input and receiving output. The is all of a kind of Chat bot prototype as this is the initial stage of development in this highly esteemed field.

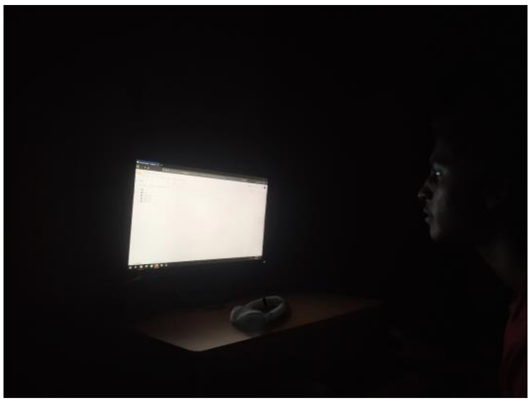


Fig 4.2 PC and earphones

Initially the audio (Speech request) has to be passed as input the system and the system responds after receiving the audio as shown in fig 4.3 and 4.4.

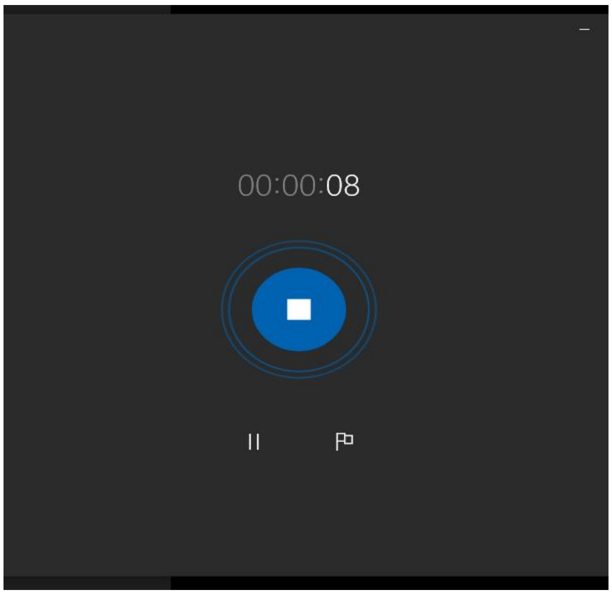
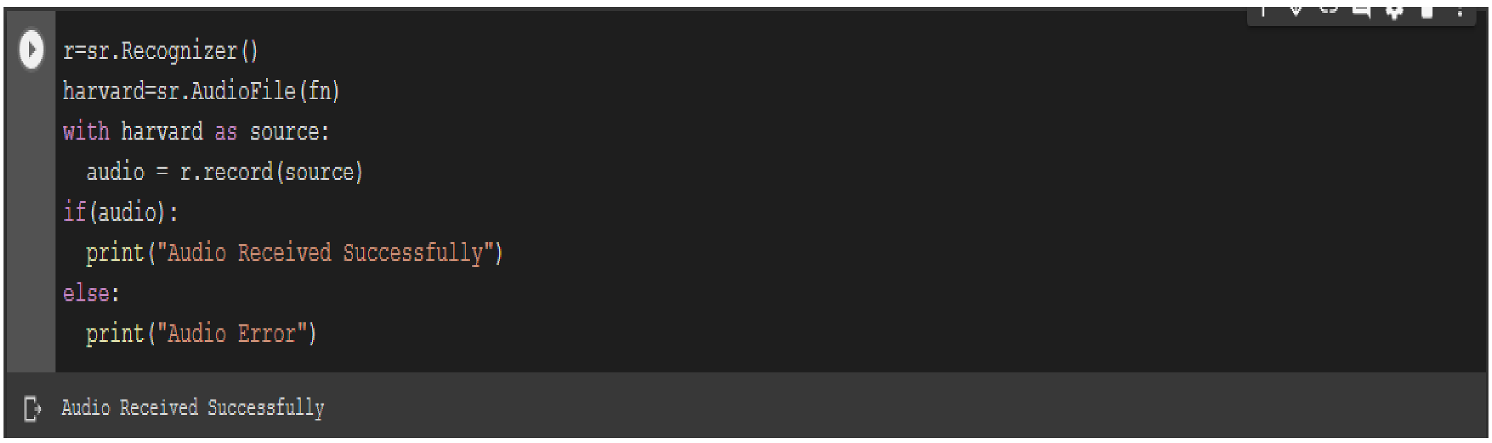


Fig 4.3 Audio input



```
r=sr.Recognizer()
harvard=sr.AudioFile(fn)
with harvard as source:
  audio = r.record(source)
if(audio):
  print("Audio Received Successfully")
else:
  print("Audio Error")
```

```
Audio Received Successfully
```

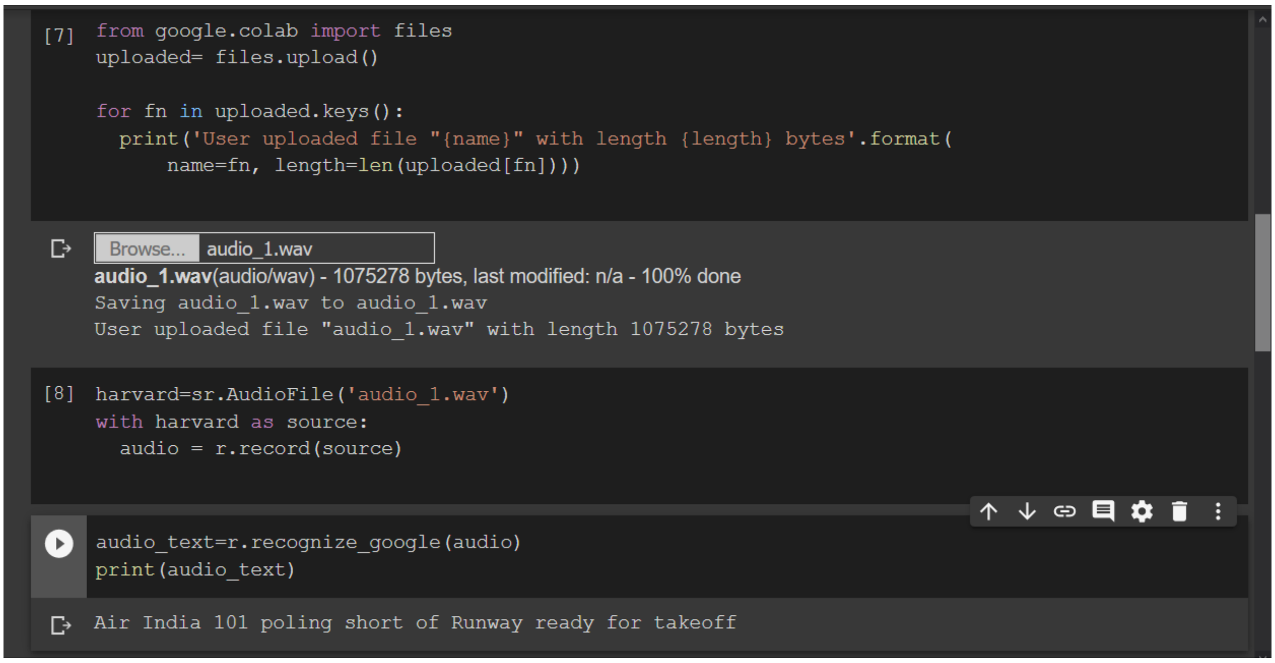
Fig 4.4 System Response after receiving audio

4.1 SPEECH REQUEST TO TEXT

One of the greatest hurdles for introducing higher levels of automation in air traffic management (ATM) is the intensive use of voice radio communication to convey air traffic control (ATC) instructions to pilots. Automatic speech recognition, which converts human speech into texts, can provide a solution to significantly reduce controllers' workloads and increases air traffic management efficiently. The first module of our system converts the pilot's request speech to text. This module can also be called as Speech recognition module. This is a vital task which takes place initially in the entire process. This module comes under Data collection because, we collect the data from pilot's request. The pilot's request is collected as audio file in wav format or it is directly obtained through hardware microphones. Then the collected audio file is converted to text, by including google speech to text package in the source code and relevant text is obtained. This is done so, in order to understand the pilot's need and intentions.

As our system is a chat bot kind of controlling system, it needs to understand the vocabulary and content in the pilot's request properly. So pilot's speech is converted to text. But during this process, noise is a concerning factor. The reason is, it may even change the context of the request from pilot, and confuses the system. So noise factor has to be considered.

Currently, several speech recognition modules require a manual adaptation to local needs caused by acoustic and language variabilities such as regional accents, phraseology deviations and local constraints. So different languages and accent are not considered as this is just a beginning of new system. So we consider common English and its standard accent. Fig 4.5 and 4.6 illustrates speech to text process in our system. We initially pass an audio file as pilot's request in standard wav format to obtain text.



```
[7] from google.colab import files
    uploaded= files.upload()

    for fn in uploaded.keys():
      print('User uploaded file "{name}" with length {length} bytes'.format(
          name=fn, length=len(uploaded[fn])))
```

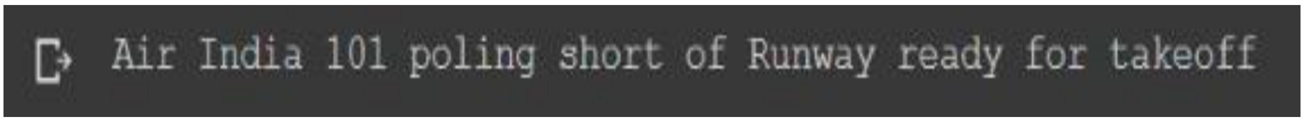
```
Browse... audio_1.wav
audio_1.wav(audio/wav) - 1075278 bytes, last modified: n/a - 100% done
Saving audio_1.wav to audio_1.wav
User uploaded file "audio_1.wav" with length 1075278 bytes
```

```
[8] harvard=sr.AudioFile('audio_1.wav')
    with harvard as source:
      audio = r.record(source)
```

```
audio_text=r.recognize_google(audio)
print(audio_text)
```

```
Air India 101 poling short of Runway ready for takeoff
```

Fig 4.5 Audio converted to text



```
Air India 101 poling short of Runway ready for takeoff
```

Fig 4.6 Converted text request.

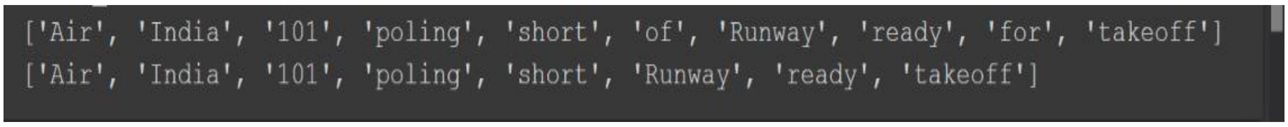
4.2 TEXT SUMMARIZATION

The module 2 of our system is “Text Summarization”. This comes under data collection and data pre-processing. This step initially gets the converted text as input. Here in this step we remove or prune unnecessary words in the statement, that a search engine has been programmed to ignore, both when indexing entries for searching and when retrieving them as the result of a search query are called as ‘stop words’ and we take only the keywords, which are termed as ‘tokens’.

LSTM text summarization is performed here. Thus the process of converting data into something a computer can understand and pre-processing only the required content (keywords) and removing the unwanted data (stop words) is known as ‘data preprocessing’. Then the keywords are collectively stored and made used for further steps.

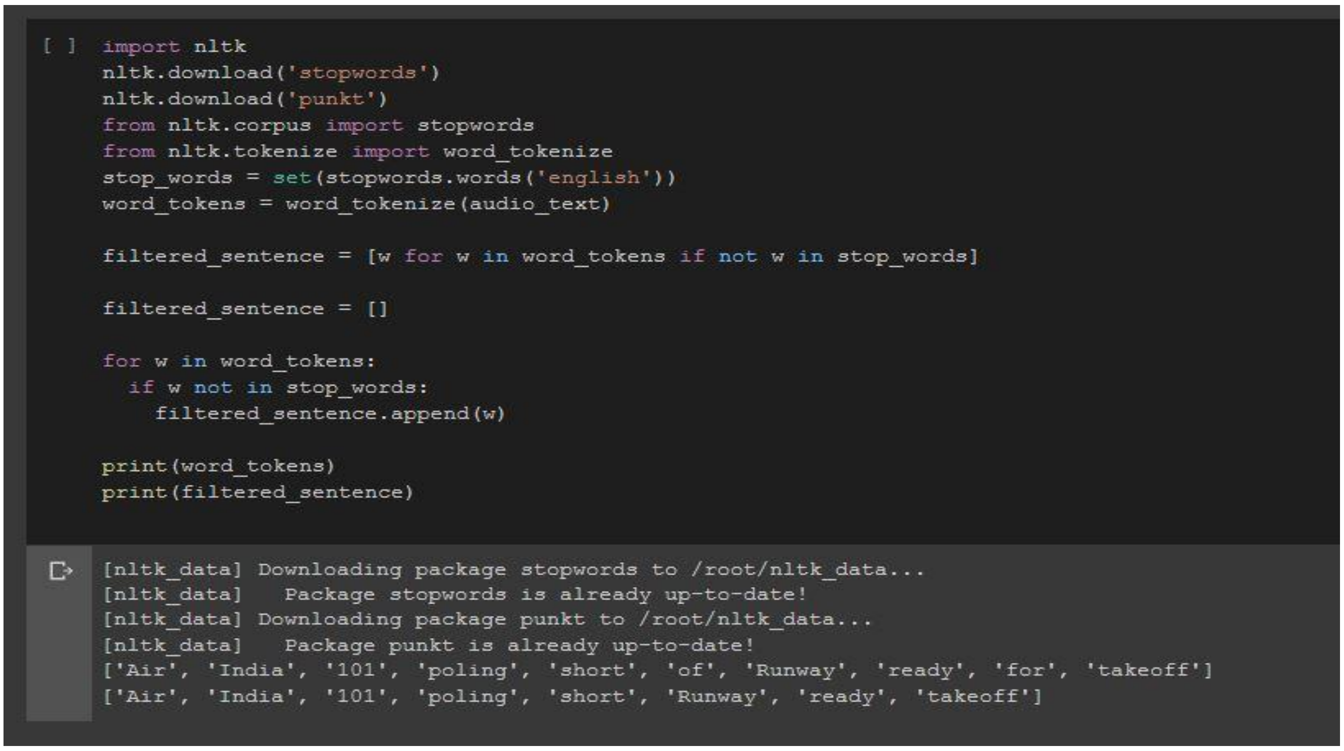
The reason behind the importance of this second module is, we generally need to know the pilot’s need just by understanding his request. So in order to make our system understand the pilot's needs, we remove stop words first and collect only the key words. Each and every keyword in pilot - controller conversation possess specific set of meanings. So tokenizing and processing keywords play an important role in our system.

Fig 4.7 and 4.8 illustrates text Summarization process. In fig 4.5 we obtained text output from audio input. That output command text is fed as input to this text Summarization module. The natural language toolkit removes unwanted stop words, and obtains only the selective keywords, which is performed in fig 4.8



```
['Air', 'India', '101', 'poling', 'short', 'of', 'Runway', 'ready', 'for', 'takeoff']
['Air', 'India', '101', 'poling', 'short', 'Runway', 'ready', 'takeoff']
```

Fig 4.7 Summarized text



```
[ ] import nltk
    nltk.download('stopwords')
    nltk.download('punkt')
    from nltk.corpus import stopwords
    from nltk.tokenize import word_tokenize
    stop_words = set(stopwords.words('english'))
    word_tokens = word_tokenize(audio_text)

    filtered_sentence = [w for w in word_tokens if not w in stop_words]

    filtered_sentence = []

    for w in word_tokens:
      if w not in stop_words:
        filtered_sentence.append(w)

    print(word_tokens)
    print(filtered_sentence)
```

```
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
['Air', 'India', '101', 'poling', 'short', 'of', 'Runway', 'ready', 'for', 'takeoff']
['Air', 'India', '101', 'poling', 'short', 'Runway', 'ready', 'takeoff']
```

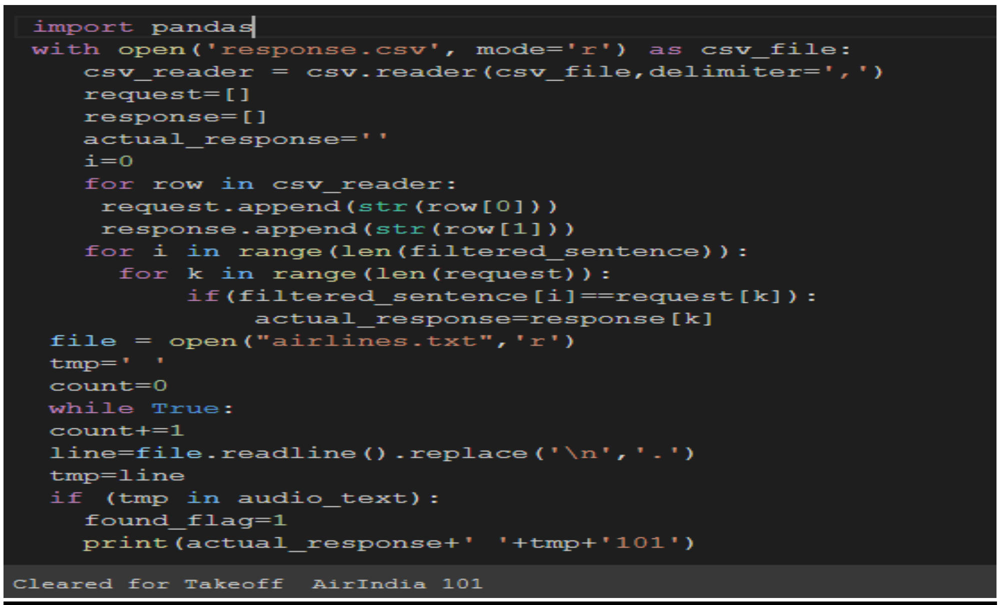
Fig 4.8 Tokenized text

4.3 RESPONSE GENERATION

The most important process takes place in this 3rd module. This is an important stage, where the system is going to generate response to the pilot, that's why its name is "Response Generation" module. Here the keywords generated from the previous module 2 are obtained. Then it undergoes comparison phase, where the keyword(s) get compared with the keywords in a csv (comma separated values) file where corresponding responses are already fed into it.

Here we compare that one generated keyword with the set of keywords in the csv file. After comparing, it generates the response for corresponding request keyword. To access CSV file contents and to perform operations in it, we use ‘pandas’ (panel-data) library to retrieve the corresponding response data from the file and from the keyword obtained from the previous module, we thereby generate suitable response. Moreover we use ‘pandas’ library for data manipulation and analysis.

Fig 4.9 shows response generation phase. We can clearly note that, the system has clearly responded to pilot’s request. It takes keyword as input request and after the comparison phase it generates the full response text. For instance it takes AI101 (flight number) and “requesting for take-off” and it generates “Cleared for takeoff” which is a partial response to the pilot. Likewise the conversation between an air traffic controller and pilot is vast. But we have chosen a specific set of basic and default conversation between them. The system generates responses for landing/take-off approach/clearance requests alone.



```
import pandas
with open('response.csv', mode='r') as csv_file:
     csv_reader = csv.reader(csv_file,delimiter=',')
     request=[]
     response=[]
     actual_response=''
     i=0
     for row in csv_reader:
      request.append(str(row[0]))
      response.append(str(row[1]))
     for i in range(len(filtered_sentence)):
       for k in range(len(request)):
           if(filtered_sentence[i]==request[k]):
               actual_response=response[k]
  file = open("airlines.txt",'r')
  tmp=' '
  count=0
  while True:
  count+=1
  line=file.readline().replace('\n','.')
  tmp=line
  if (tmp in audio_text):
    found_flag=1
    print(actual_response+' '+tmp+'101')
```

```
Cleared for Takeoff  AirIndia 101
```

Fig 4.9 Response Generation

4.3.1 FLIGHT INFORMATION INCLUSION

This sub-module is named “Flight information inclusion”. The main use of this sub-module is to add flight details alongside the response command that is generated in the previous main-module. The reason for adding flight details is to identify their own responses for pilots as, not only one aircraft is going to use the aerodrome for takeoff/landing. The controller has to manoeuvre many aircrafts. So while responding, they use the flight number of that aircraft with their response, sometimes from/to aerodrome also. We retrieve the flight details, in the beginning itself and add additional details, along with the response.

Flight information is obtained from flight plan that is given by respective pilots to controllers that holds all the vital information about the aircrafts which includes flight name, number, variant, from/to aerodrome etc. Every aircraft will hold its own unique detail of the day. So retrieving the flight detail (aircraft number) is important.

Flight detail is retrieved from CSV file. We create a flight plan model and store it in a file. So when the pilot requests, they will surely mention the aircraft ID (flight number) which is a mandatory rule. This flight number is retrieved from that request initially after data pre-processing stage. Then the flight number is compared with the data in the flight plan model. After comparing, flight the necessary flight details are obtained.

The obtained flight details are stored and after processing of other details, such as response generation, all these details are added or appended or concatenated with the response and proper response is generated for the pilot’s response.

4.4 TEXT TO PILOT RESPONSE

The last and final module of the process is “Text to Pilot response” module. After successful completion of obtaining response text for pilot’s request by executing all these above modules, it has to be converted to speech (audio) format. After generating audio response, it has to be sent to the pilot, who waits for response from controller as shown in fig 4.10. The request-response conversation has to be quick and spontaneous.

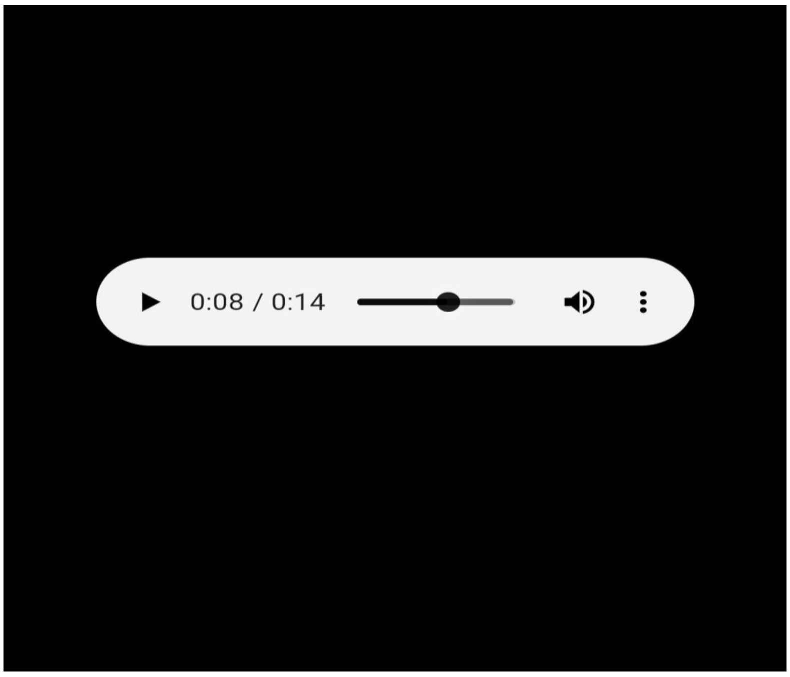


Fig 4.10 Audio response

4.5 SUMMARY

These are the 4 modules make up our automatic Controller Bot in Air Traffic Control System. It thereby helps the controller in maneuvering the aircraft safely under his/her surveillance and reduces their mental stress in a great manner. This automatic controller bot in Air Traffic Control System has to be spontaneous in action and also it will ensure that correct response is generated to pilots, as per their request. System is implemented as described in this chapter. And also the interconnection of the modules can be understood from this chapter. The working of each module is detailed along with its usage. The work of each module appears to be sequential. The minimum time for the entire process to complete after receiving request is expected to range from 30 seconds to 1 minute. This proves the spontaneity of the proposed controller bot.