Report - Automatic Speech Recognition

- Course: Human Computer Interaction
- Student Name: Duan Tingting
- Student ID: 2152402
- Tutor: Deng Hao

1. Brief introduction

This is a program written in Python using the PyQt5 library to create a GUI-based voice assistant. The program uses the speech_recognition library to recognize speech and then performs various tasks based on the user's voice commands.

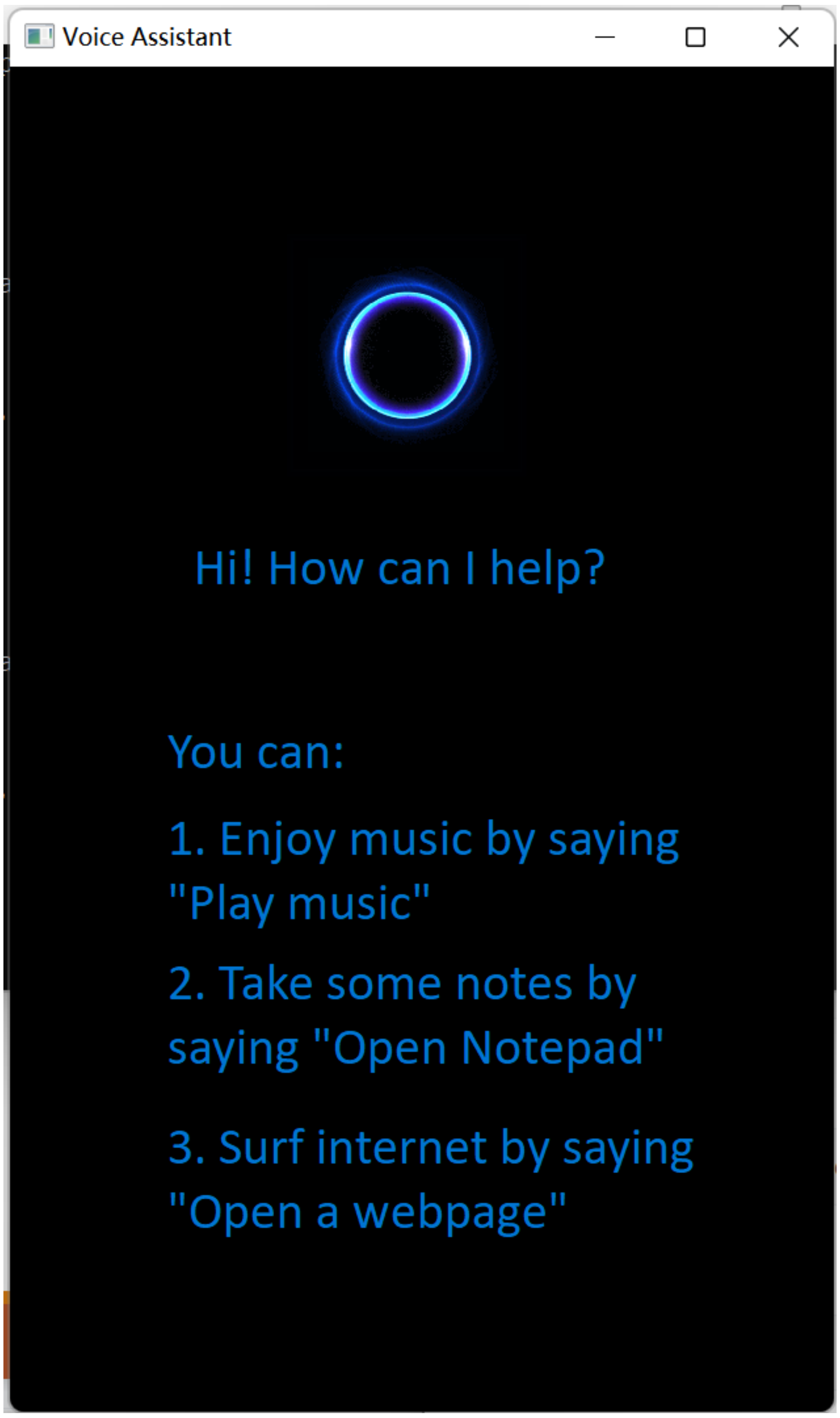
2. The modifications to GUI

The program creates a GUI window using the Qt framework and sets up several QLabel widgets to display various messages to the user. The program also sets up a QMovie widget to display an animated GIF image of a voice assistant.

Firstly, I have made adjustments to the page size in order to enhance usability for the end user.

Secondly, I have made modifications to the position and size of text in order to optimize clarity and suitability for display purposes.

Lastly, I have implemented a functionality that allows for the opening of web pages.


Voice Assistant

Hi! How can I help?

You can:

1. Enjoy music by saying "Play music"

2. Take some notes by saying "Open Notepad"

3. Surf internet by saying "Open a webpage"

3. The explanation of codes

3.1 class Ui_MainWindow(object)

The Ui_MainWindow class defined in the code provides the design of the main window. It sets up the various QLabel and QMovie widgets used in the program and defines their positions and sizes. The retranslateUi() method of the Ui_MainWindow class sets the text of the various QLabel widgets to display the appropriate messages to the user.

3.2 class myWindow(QtWidgets.QMainWindow)

The program's main window class is called myWindow and inherits from the QMainWindow class provided by PyQt5. The class defines a showEvent() method that is called when the window is shown. The method sets up an instance variable called myThread to None.

3.3 class MyThread(threading.Thread)

`MyThread` extends the `Thread` class from the `threading` module. The class also imports the `pyaudio`, `speech_recognition`, `win32api`, and `webbrowser` modules.

The `run` method overrides the `run` method of the `Thread` class and is called when the thread is started. The method continuously listens to the microphone and transcribes the speech using either Google Cloud Speech or Sphinx, depending on whether the device is connected to the internet. The recognized command is then executed by calling the `execute_command` method.

The `execute_command` method takes a recognized command as input and executes an action based on the keyword(s) found in the command. For example, if the command contains the word "music", it opens Windows Media Player. If the command contains the word "notepad", it opens Notepad. If the command contains the word "webpage", it opens Google in the default web browser.

The `recognize_speech_from_mic` method takes a `Recognizer` object and a `Microphone` object as input and transcribes speech from the microphone. If the device is connected to the internet, it uses Google Cloud Speech to transcribe the speech. If the device is not connected to the internet, it uses Sphinx. The method returns a dictionary with three keys: `success`, `error`, and `transcription`. The `success` key is a boolean indicating whether or not the speech recognition was successful. The `error` key is a string containing an error message if an error occurred. The `transcription` key is a string containing the transcribed text.

The `stop` method sets the `_stop` attribute to `True`, which stops the thread from running.

4. Accuracy improvements

To improve the accuracy of speech recognition, I made the following improvements:

1. Since the speech recognition accuracy of Sphinx was low, I replaced the speech recognition model with the **Google Cloud Speech** model, which significantly improved the recognition accuracy. It is important to ensure a good VPN connection before using the recognize_google() function. However, if there is no network, I will use the Sphinx recognition as **a backup solution** to enhance the robustness of the program.

```
            if isNetConnected():
               print("Google Recognizing\n")
                response["transcription"]=recognizer.recognize_google(audio)
                if response["transcription"]=="":
                    print("NULL")
            else:
                print("Sphinx Recognizing\n")
                response["transcription"] = recognizer.recognize_sphinx(audio)
```

2. I changed the recognized content from sentences to keywords and added similar-sounding content to the recognition database, increasing the probability of successful recognition.

```
    def execute_command(self, command):
        music_keywords=["music", "using", "you may", "moving", "you mean the",
"meaning", "you think"]
        notepad_keywords=["notepad", "no pad", "no impact", "no that", "note
pad"]
        webpage_keywords=["web page", "webpage", "web", "lap pads"]

        if any(keyword in command for keyword in music_keywords):
            win32api.ShellExecute(0, 'open', 'wmplayer.exe', '', '', 1)
        elif any(keyword in command for keyword in notepad_keywords):
            win32api.ShellExecute(0, 'open', 'notepad.exe', '', '', 1)
        elif any(keyword in command for keyword in webpage_keywords):
            webbrowser.open("https://www.google.com")
```

3. Adjust the recognizer sensitivity to ambient noise and record audio.

```
audio=recognizer.record(source,duration=5)
```