Elliott Rose

COSC 625

Prof. Huang

03/23/20

Mid-Term Progress Report: Automated Research Assistant

This project was intended to assist in research by accepting voice commands, and then reading the returned command back to the user. The project would use open source libraries for Python to combine the necessary technologies for the assistant. The requirements for the system were that the system would listen to the user's query, search for an answer, read the information retrieved back, and then take commands to search the returned information for keywords. The following paper will describe the current state of the project and the challenges encountered, as well as what I have done and plan to do to deal with these challenges.

To make the development of this project go more smoothly, I chose to separate the functional components into separate pieces for their development. This required separating the speaking engine, the listener, and the query answering function into separate components. I did this to keep coupling within the system to a minimum, so that as the project progresses, components could be easily modified or changed as requirements developed or changed. This allowed for easier development cycles as I was also able to more easily concentrate on the specific component, and only worry about how it should act. Tying everything together later would make dealing with issues of separating out processes using threading or multiprocessing easier as well.

Firstly, I build the speaking portion of the app, using the Pyttsx3 library for Python. This module is considered the best open source speaking engine for Python. This engine allows the

user control over the speaker, allowing to interrupt an utterance after it has begun. This is a key point and proved to be difficult to deal with even with the built-in methods in Pyttsx3. The interrupt method that they have uses a trigger on a word count; this meaning that after a certain number of words are spoken, the interrupt method is triggered, and the speaker stops. I believe the method is written this way because the command line is not available until the speaker finishes, making it impossible to call the method while it's running. This would not work for the system, so I had to utilize a thread in python to keep access to the command line. This allowed me to use a while loop with an awaiting input to trigger the interrupt method. This took some time to get right, but not it works well.

There are still some problems in using Pyttsx3 though. The engine does not throw the proper callback when it is finished speaking, when using threading. Because of this, when the speaker is finished, the thread has to be manually stopped to get back to the listener. This is not a critical problem, as there a many usable workarounds to deal with this issue, but I would like to see if I can find another way to trigger the end of speech callback. As a temporary fix for this issue, I have decided to use Google Text To Speech to fill in the assistant's consistent responses. So, responses like, "What can I do for you?" or, "Ok, searching Wikipedia" are now spoken by a different speech engine. This is mainly because GTTS works by converting text to speech into mp3s for saving, storing, and playing. This is precisely the reason that GTTS is a poor choice for this assistant, because the engine takes much more time to create a response, as it has to save the whole file before responding. The benefit is that the engine finishes and allows the code to continue.

The second portion of the system that I worked on was the query system. For this project, I wanted to have assistant be able to search Wikipedia, read notes on the local computer, and

answer some natural language questions. For the natural language questions, I found the START systems by MIT to be the best option. It is a website for a project they are running that takes in many different kinds of natural language queries(“Who is the president of the United States?”, “What is is the world’s tallest mountain?”, and “What is twenty two times fourty four?”) and attempts to answer them. The system is fairly reliable but does take a bit of time to return an answer. To take advantage of the START system and Wikipedia, I used the Requests module with Beautifulsoup for web scaping. Now the system returns answers converted into text strings for the speaker to read. The notes portion just took accessing whatever note files are stored in the current directory or subdirectory and then reading them as strings.

The final portion of the system is the listener. The listener has shown to be the most problematic of all the systems. I have tried to speech-to-text conversion packages for Python, those being: Google Speech, and Carnegie Mellon University’s PocketSphinx program. Of the two, I have found that the Google Speech is really the only usable open-source speech recognition yet. The PocketSphinx was incredibly inaccurate, making it essentially useless as it was not dependable. This was disconcerting because PocketSphinx is the only open-source speech to text conversion that does not require an internet connection. This would help to make results faster and avoid complications due to poor internet connections. However, at this time I am still attempting to tune the parameters of PocketSphinx to get a reliable result, having not yet produced one.

On the other hand, Google Speech does allow a limited number of queries per day for free and does not require an API key. So, this has been my go-to at the projects current state. Overall the services have been a bit problematic, but useful. Google’s service often has delays in recognizing speech and is also inaccurate sometimes but is mostly reliable. The real problem has

been finding a way have the assistant listen while it is speaking. A few problems arise because of this; The first is that if a headphone with a mic, or Bluetooth headset are not used, the speaker speaks through the computer speakers, and thus hears its own speech. So, at this moment, without paying for a subscription service to Google which offers identification of separate speakers, it will only function when using some sort of headset. That being said, even this doesn't seem to work perfectly, as the listener still sometimes picks up the assistant's speech. Further, at this moment I have not been able to get the listener to reliably run while the speaker is running. It is buggy, and sometimes it will run fine, and others it will hang and the program sticks. I'm still searching for a solution. In the meantime, the program runs reliably when accepting input from the keyboard while the assistant is speaking. The user can successfully stop the program or search for a keyword within the returned text, and the speaker will stop and return the searched portion of the answer.

For the coming months in the project, I tackle the final issues of the project, and build a simple user interface. I would like to conquer retrieving the callback when the speaker finishes, and iron interrupting the speaker with the speech recognition when using a headphone/mic setup, or Bluetooth. I am not sure that I will be able to find my way around correcting the complications of the listener hearing the speaker when speaking, so if I cannot fix this portion, I can default back to a speaker only interrupted by text input. For the user interface, I hope to make something that I can build for Android, as I see this software as being most useful from a mobile device. Once these issues are all addressed and the software has been thoroughly tested, it will be a finished project for this assignment, and the early stages of a much bigger project.

Bibliography

Sommerville, Ian. Software Engineering. 10th ed., Pearson Education Limited, 2016.

“SpeechRecognition.” PyPI, pypi.org/project/SpeechRecognition/.