EPICS Spring 2024 Documentation

Project Overview

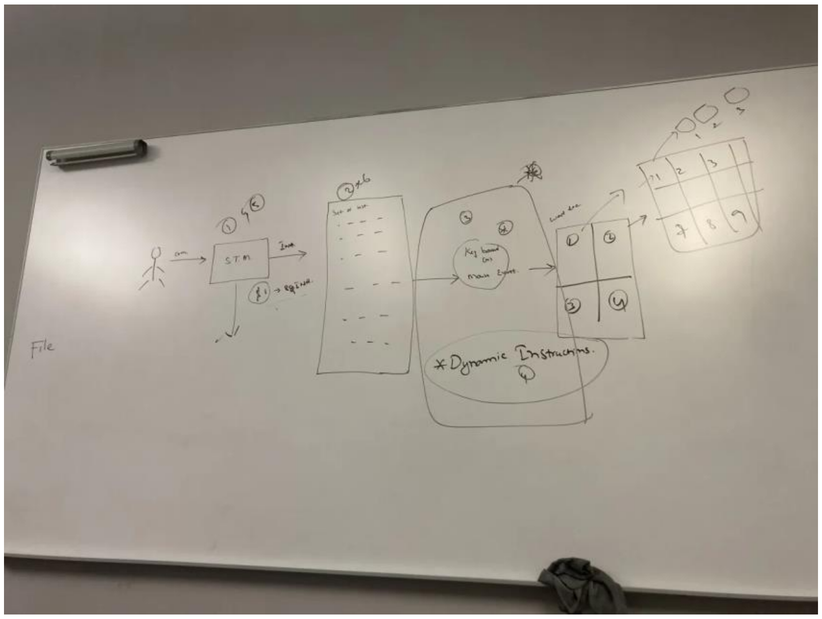
Kalasalingam University in south Tamilnadu has requested assistance from UTD in creating a speech recognition software to help people with spinal cord injuries access and operate computers more easily. The software should translate user voice commands into keyboard and mouse events, with a potential future integration with eye-tracking software. The entire project Kalasalingam is trying to build has two parts of which we are only concerned with one: the speech recognition.

The two parts are the eye tracking module and the speech recognition module (we are only concerned with the speech recognition). The original request from our partner was to create an interface where the screen is split up into a 3x3 grid from which the user can keep selecting into a grid box to eventually press something using their voice. However, since the eye tracking would already be able to navigate the screen, we thought it was redundant and so, we changed the goal. The new goal was to be able to execute commands using speech on the screen. This would take some time so we need to start small and work our way up.

Thus, the immediate goal for this semester is to develop a functional speech recognition prototype that can execute specific commands on Notepad. However, the project faces budget constraints, limiting the options for speech recognition models as most commercial ones charge based on audio time or requests made. Considering the potential high usage and costs, the project is left with only open-source models for speech-to-text conversion.

Minimum Viable Product (MVP) | Functional Requirements

This diagram shows the original idea for how the application was going to work.



The original application design was intended to perform operations on the Windows Notepad Application. It needed to be able to do things like write, copy, paste, and select text in the app.

The high-level workflow was as follows:

1) The client speaks.
2) The Speech-to-Text (STT) Model receives the speech and converts it to text.
3) The text is processed and sent to the command dictionary.
4) The Command Dictionary maps the text to a specific event.
5) The event is then activated.

While we adjusted as the project progressed, this was the foundational concept.

Design Process

Initially for command processing, we planned to use a Priority Queue to track commands and skip all commands relevant to an invalid command. However, we decided to take one command at a time instead because it caused a lot more issues if a command was incorrect, also it proved somewhat unnecessary for our current goals. If a command is not an exact match, it should be closely matched to the next best command; if the command is too far off then we just tell the user it is an invalid command.

We also considered implementing a termination command and using a Python framework to create a desktop application. The application was intended to use the Microsoft Grid System to divide the screen into a grid and potentially use Optical Character Recognition (OCR) to convert images to text. The application was also designed to support keyboard shortcuts for actions. For example, if some text is selected, the user can say "bold" to trigger the CTRL + B shortcut and bold the text.

For the event implementation we decided to use the Python Pyautogui Library as it was the most straightforward when it came to implementing keyboard and mouse events.

We implemented a basic front-end using Tkinter for the user to input their own activation keyword.

Text Stack & 3rd Party Integrations

To begin, there really isn't any API's or other software we integrated besides using the Vosk Speech model if you want to count that.

As for our tech stack, the entire code runs on pure python.

Work Distribution

Having said that, the project team was then divided into three specialized groups:

1) **Model Identification Team**: This team was responsible for identifying the appropriate model for our speech-to-text application.
2) **Command Dictionary Team**: This team was tasked with researching and creating a comprehensive command dictionary that encompassed all necessary commands for the project. Essentially, the command given by the model would map to this dictionary and execute the necessary function
3) **Event Implementation Team**: This team was responsible for implementing all the functions within the command dictionary. This was the code that took care of keyboard and mouse movements

As aforementioned, a separate group was initially planned to develop a system that divided the screen into grids, facilitating client navigation across different screen areas. However, this became redundant as an international team was concurrently developing a similar system using computer vision models. This is the reason the work was as such.

Moving forward having finished those tasks, we needed to focus on new problems that arose during development.

Later in the project, the team was restructured into two subgroups:

1) **Speech-to-Command Mapping Team**: This team developed a system that received client speech and properly mapped it to a command within the dictionary. For example, the word "waste" would be mapped to "paste", accounting for accents, hardware limitations, and model inaccuracies. This team utilized the Python FuzzyWuzzy library for this task.
2) **Model Optimization Team**: This team was tasked with optimizing the original model to meet our standards or identifying a suitable replacement. Further details on this process are provided in a subsequent section of this document.

Model Selection

Given the project's requirement for Speech Recognition, it was crucial to identify a model that met our specific needs. Our initial choices included:

- Whisper
- Deep Speech (Mozilla)
- Speech Recognition (Python Library)
- Google Speech To Text
- Vosk

During testing, we found that these models, along with others not listed, either required excessive computational power, exceeded our budget, lacked accuracy, or had lengthy processing times. We also considered how each model processed text, i.e., whether it converted the text into a file for processing.

Initially, we selected the Python Speech Recognition Library, but it presented several issues. We used two sub-models within it for online and offline access, but the online model had slow response times, and the offline model lacked accuracy. The model also occasionally failed to pick up audio, complicating debugging and testing.

We ultimately switched to the Vosk Giga Speech model, which offered satisfactory response times and accurate transcriptions. It was relatively quick to set up and easy to implement. The primary drawback was the model's large size and its lack of punctuation and capitalization features.

Dictionary and Keyboard Mouse

Implementing the dictionary and keyboard and mouse events was fairly straightforward. It was just a matter of creating a python dictionary whose keys were the actual command to be mapped to and the values be the function call of the event to be executed as you will see in “CommandDict.py”.

Moreover on the function calls, you will find them in “Events2.py”. Here each function makes use of the *pyautogui* python library to navigate the screen and execute commands. If you'd like to know more about this library, their website has great documentation.

That being said, there are certain commands that need more than just these two parts to execute properly. Those commands that need more code are the 1. activate command (wake word) 2. terminate 3. start/stop dictate 4. mute/unmute. The code relate to these commands can be found in “CommandExecutor.py”. To understand this better, let's look closer at the code.

In command executor, its starts with the initial variables and its initial states. The mute and unmute is taken care of using the setMute() function which just toggles the variable controlling command execute in the execute_command(). Start and stop dictating is toggled in the toggle_dictation_mode() function in the same manner as mute and unmute. The actual dictating would be using the type_text function. The biggest part of this file is the execute_command() function. This function first checks for dictation mode and mute mode and if both are false, then it should process the command. The command is processed to match to a command using the *fuzzywuzzy* library, then it will execute the closest match, accessing the dictionary. As for the activate and terminate command execution, it is taken care of in the Speech recognition file/part of code; please refer to that section of this documentation to learn more.

Goals For Next Semester

- **Improve Model**: While the current model performs its job adequately, if possible, we should be able to find or improve the model to take on a wider variety of voices, this includes different accents as well.
- **Improve Speech to Command Mapping:** While we did implement the FuzzyWuzzy library to map the speech string to the command within the command dictionary, there seems to be some inconsistencies with the library, i.e. it maps the word "Random" to the command "Undo". While we can of course increase the confidence value, I believe that possibly finding a better library to perform this task will be better.
- **Packaging and UI**: The next team should aim to package this program and pair it with an interactive user interface making it easier for clients to use.
- **Expand Scope of Project:** As of right now this project is mainly used for the Windows Notepad application. The next group should aim to allow the program to handle more tasks such as browsing the internet through a web browser or controlling parts of your local machine through text commands.

Instructions for setting up the environment:

- Use any version of Python 3.11
- Download 40M and 2.3 GB Vosk models from https://alphacephei.com/vosk/models
- Libraries to pip install: fuzzywuzzy, pyautogui, pyaudio, SpeechRecognition
- Or run pip install –r requirements.txt to install everything at once.

Requirements.txt - fuzzywuzzy pillow PyAudio PyAutoGUI PyGetWindow PyMsgBox requests SpeechRecognition typing_extensions urllib3