International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 10 Issue 08, August-2021

Voice Assist and Control through Hidden Markov Model [HMM] and Natural Language Processing [NLP]

An Application to Recognize, Interpret and Execute Speech Commands

Mrs. Pooja B S
Assistant Professor: Dept. of Computer Science & Engineering
BNM Institute of Technology
Bengaluru, India

***Abstract*— The absence of a system where spoken commands to a computer are standardized, or at least confined to a dialect, leads to increased complexity in providing speech support. To combat this problem, we propose to develop: VAAC - Voice Assist and Control [pronounced /väk/]: an application to recognize, interpret and execute speech commands across other applications and programs. To implement such a system, techniques of Speech Recognition through HMM models and Natural Language Processing: Entity Extraction by Word Matching & Co-reference and Anaphora Resolution is proposed. To enable automated learning, Machine Learning Techniques of Language Model training, Acoustic Model training and Phonetic model generation, which improve speech recognition accuracy are used.**

Keywords*— *Hidden Markov Model, Natural Language Processing, Entity Extraction, Anaphora Resolution, Acoustic Model Training, Phonetic Model generation

I. INTRODUCTION

Interacting with computers by voice is a complex task involving speech recognition and natural language processing. In many cases, it makes sense to provide support only for a certain set of commands: this is known as command and control. However, such a system to interpret and interact through voice commands would provide great ease of access through command and control.

Command and Control applications are concerned with providing the user of these systems, the means to control items within their environment with voice commands appropriate to the domain. The appliance of command and control technology may manifest itself in the control of user interface menus in personal computing desktop applications or the control of large scale mechanical or electronic and computing devices. Command and Control to develop a Voice User Interface for programming assistance is implemented. A VUI is the interface to any speech application which relies on technologies like text-to-speech, speech-to-text, Natural Language Processing and Entity Extraction to develop a ubiquitous programming assistant.

Many command & control programs do exist, for example in Home automation, automobile command and military applications. These are built for very specific applications in platform specific environments, with large amounts of development, and training time. These programs are not scalable and maintainable by the general community. Also, they do not provide sufficient accuracy when support for a wider range of commands is necessary.

Most voice assistants do not provide interfaces into other applications. There is no single program which can take vocal input and execute commands in the format of an input device. There is a lack of a dialect for spoken commands to a computer. We address this problem, and propose *VAAC*, which fundamentally is the equivalent of an input device which takes speech as input and executes commands as output.

II. SYSTEM DESIGN

A. *Proposed Model*

In the proposed approach of the implementation of *VAAC*, it uses PocketSphinx as the speech recognition engine. PocketSphinx is an open source toolkit for speech recognition. The system requires SphinxBase and SphinxTrain libraries for setting up and training the PocketSphinx model. It consists of a framework that eases the creation of a PocketSphinx model for an individual speaker to perform command and control, and provides a simple interface to make use of this model to perform basic tasks in other applications.

The system on initial deployment can support applications such as: *Mozilla Firefox, Visual Studio Code, Gedit, Gnome-Terminal, Nautilus* and will work on *Ubuntu 18.04* with the *Gnome3* shell. It works by matching the words in the command to a list of standard commands supported by the application. These standard commands will then be translated into different types of keyboard events before being sent to the target application via *xdotool* and other utilities.

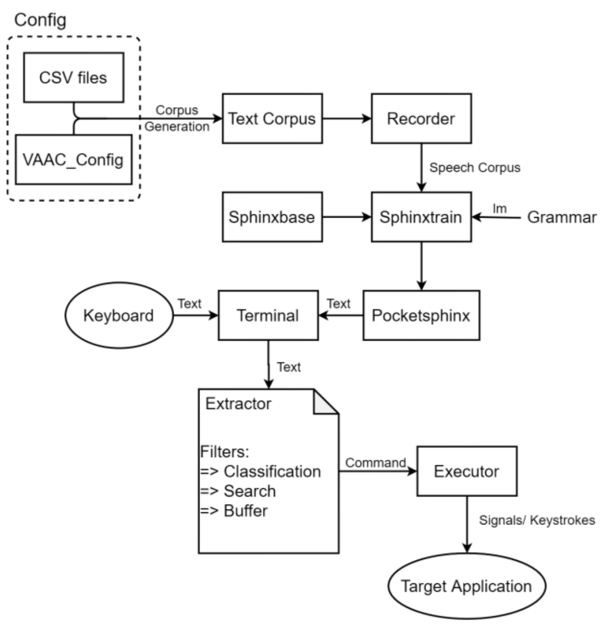


Fig 1. Proposed Model for VAAC

The system aims at providing sufficient support to complement actions performed using other input devices and enhancing device accessibility by incorporating it in all computing environments. Following are the vital components utilized to implement the *VAAC* system.

i Pocketsphinx: Speech recognition can be useful in many scenarios such as personal assistant bots, dictation, voice command-based control systems, audio transcriptions, quick notes with audio support, voice-based authentication, etc. PocketSphinx is a library that depends on another library called SphinxBase which provides common functionality across all CMUSphinx projects. The CMUSphinx toolkit is a speech recognition toolkit with various tools used to build speech applications. CMUSphinx contains several packages for different tasks and applications.

ii *xdotool*: It lets to programmatically (or manually) simulate keyboard input and mouse activity, move, and resize windows, etc. It does this using X11's XTEST extension and other XLib functions. There is some support for Extended Window Manager Hints. Aliases exist for ‘alt’, ‘ctrl’, ‘shift’, ‘super’, and ‘meta’ which all map to Foo_L, such as Alt_L and Control_L, etc. In cases where the keyboard does not actually have the key that want to type, *xdotool* will automatically find an unused keycode and use that to type the key. ‘Command Chaining’ allows the consumption of pending arguments or until a new *xdotool* command is seen, because no *xdotool* commands are valid keystrokes.

iii NCurses (New Curses): It is a programming library that provides an application programming interface (API) that allows the programmer to write text-based user interfaces in a terminal-independent manner. It is a toolkit for developing GUI application software that runs under a terminal emulator. It also optimizes screen changes, in order to reduce the latency experienced when using remote shells.

iv *wmctrl*: It is a tool or rather a command that can be used to interact with an X Window manager that is compatible with the EWMH/NetWM specification. *wmctrl* can query the window manager for information, and request for certain window management actions to be taken. *wmctrl* is controlled entirely by its command line arguments. These command line arguments are used to specify the action to be performed (with options that modify behavior) and any arguments that might be needed to perform these actions. Only one action can be executed with the invocation of the *wmctrl* command.

Initially, the user needs to set up a few resources before making use of the text corpus facilitated by the application to make recordings of the corpus. The recordings are made using internal or external microphones. The grammar file of the recordings is generated, and the model is set up to train the *VAAC* corpus. Use of short phrases is recommended although the phrases need not be exact commands. The recordings try to balance the frequency of words so that each word accounts to equal weights for their accurate identification.

Notably, the corpus contains a list of commands, specific to the application supported by the system. Once the model is trained, the application is ready for use. It expects vocal inputs which need to be commands within the trained corpus that are then converted to raw text. From the obtained text, entities like command and the target application are extracted. To ensure the exact matching of the entities, word-matching is used. Efficient algorithms are utilized over sorted command lists for optimum searching to ensure results in real time.

Once the application starts running, the user has the option to either give vocal inputs or type the commands in the command prompt as a failsafe. The commands are placed in a buffer from which the target application is extracted first. This is proceeded by applying certain filters to the text to classify and search the matching command.

The obtained entities include type of command, keystroke and the target application which are passed to the executor module that generates the relevant keystrokes to facilitate the execution of the command over the respective target application. Hence the executable command is generated from the entities by matching the keyboard events corresponding to the command entity.

The application can be left running in the background for the user to use it as an assistant while using the supported applications.

The user can also type ‘*help’* followed by the command name to get the documentation on the usage of that respective command. To terminate the execution of *VAAC* in background, the user can use ‘*exit’* command. In case of an ambiguity, the application makes sure not to disturb the flow of the user by not executing any junk vocal input detected by it. The user can also view the commands detected by the application immediately on the terminal after they speak and can also check the log file to evaluate the accuracy.

B. *Data Flow Diagrams*

i. DFD Level 0: The input device, generally the microphone or keyboard provide the initial input to the *VAAC* system which identifies signals and keystrokes and transmits it to the application module to perform necessary operations.

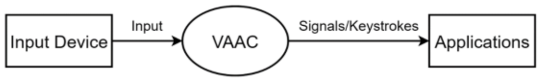


Fig. 2. Data Flow Diagram Level 0

ii DFD Level 1: The process is represented in a higher detail. The input from the microphone and keyboard in the form of voice and text is sent to the terminal, which then converts these signals into textual format and sends it to the extractor module. The extractor module classifies the text into entities such as types of commands, key strokes and target applications. This is then passed through another extractor module where signals and keystrokes entered are identified and further sent to the application module to perform necessary operations.

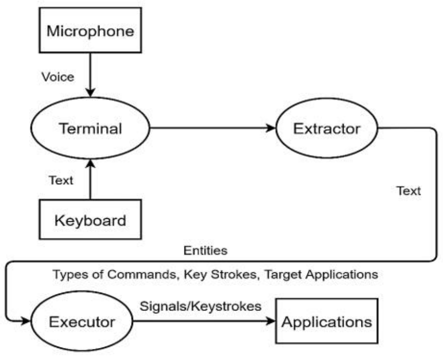


Fig. 3. Data Flow Diagram Level 1

iii DFD Level 2: The inputs in the form of text and voice are directed to a buffer, this module converts the input into textual form which is sent to identify the target application and to a filter module where the test is classified and searched for commands. These commands are passed onto an executor module and finally the signals from this module is channeled to the application module to perform the necessary operations.

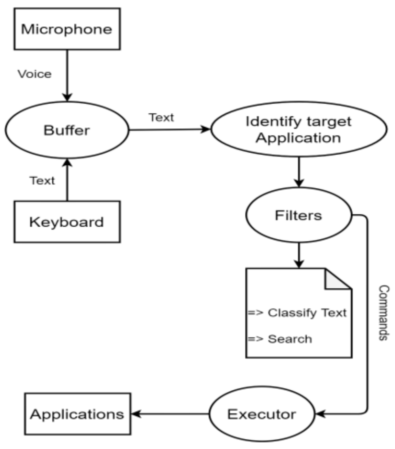


Fig. 4. Data Flow Diagram Level 2

III. Implementation

This is the logical conclusion, after evaluating, deciding, visioning, planning, applying for funds and finding the financial resources of a project. Technical implementation is one part of executing a project. This section discusses about the various algorithms that are used, and the reason behind using them.

This section elaborates on the functioning and the operations of the important modules that *VAAC* is comprised of, the algorithms and approach used in implementation are also defined in this section. Following are the modules implemented -

A. *Recorder Module*

When the recorder module is initiated, the recorder.py script prompts a phrase from the corpus and records it into a file in the recordings folder. If the phrase recorded is a single word, it will be stored under the '*recordings/{word}/{word}_{n}.wav*' extension. Else if the phrase recorded is a line from the corpus, it will be stored under the '*recordings/{corpus_name}/{line_number}_{n}.wav*' extension, where 'n' is the nth recording of the phrase recorded by the module. To initiate recording, the user is expected to read aloud the prompted phrase into an audio input device such as a microphone. Relevant options are displayed to start recording, stop recording, store recording, re-record without storing and to exit. If one makes any mistakes when recording it can be skipped using the re-record while storing option. However, if it is saved by mistake and one wants to redo it, simply delete the recording, and run recorder again. It will automatically search for missing files and prompt the corresponding phrase.

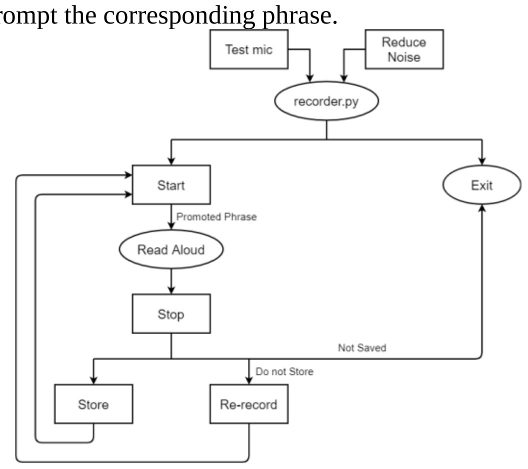


Fig. 5. Recorder Module

B. *Setup Module*

The Setup Module has the ability to unzip and extract the contents of '*TAR****.tgz*' file downloaded from *lmtool* into '*vaac_model/etc*' extension. The *lmtool* builds a consistent set of lexical and language model files for decoders. The target decoders are the Sphinx family, though any system that can read ARPA-format files can use them. Currently *lmtool* is configured for the English language.

The input to *lmtool* is a file containing the corpus which has to be trained. Notably, a corpus file needs to be generated consisting of all sentences that needs to be recognized by the decoder. The sentences should be one to a line without any punctuations. It also renames the files to have a prefix *vaac_model*. This is to easily identify and streamline the process.

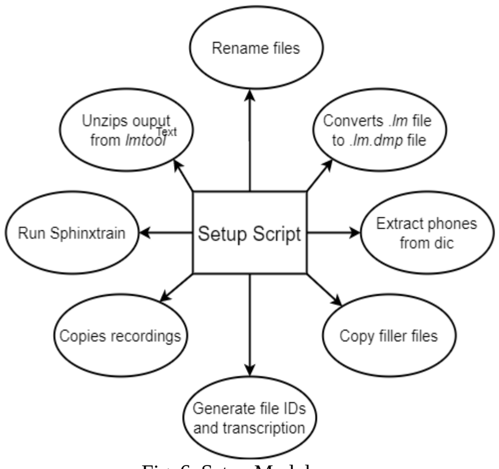


Fig. 6. Setup Module

C. Terminal Module

The terminal.py script provides an interface to the client to use VAAC. One can type into this terminal, or speak into it. Because of inherent inaccuracies in speech recognition, the terminal will not prompt messages for commands that do not make 'sense'. That means, if *VAAC* cannot understand a certain command, it will not prompt errors in the terminal. However, such errors are recorded in the logs.

This module acts as an interface between the target application and the user. It allows the users to speak their commands that are entered into the terminal, to search for commands that available and specific to that application or to list out generic commands that can be executed on any application window.

The terminal normally accepts simple, natural language commands containing words already recorded by the user during recordings. This means that the quality of recordings plays a significant role in identifying the words. In case the words spoken or detected are not a part of the corpus, similar sounding words from the corpus are referenced.

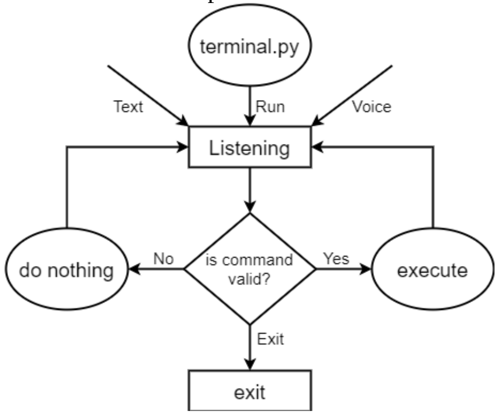


Fig. 7. Terminal Module

D. Extractor Module

The extractor module comprises of the Extractor class. This Extractor class provides various methods to extract and write words from the commands to run them. There are certain filters inside this, these filter methods return the matched command.

The extractor module assumes the following default parameters:

i Browser: Mozilla Firefox.
ii Text editor: Gedit.
iii IDE: Visual Studio Code.
iv Terminal: Gnome-terminal.
v Files: Nautilus.

The extractor module accepts the input from the terminal, evaluates it and determines the target application. It also detects the command that needs to be performed in that application.

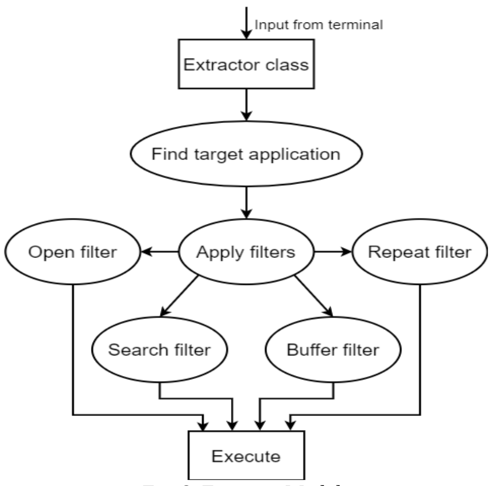


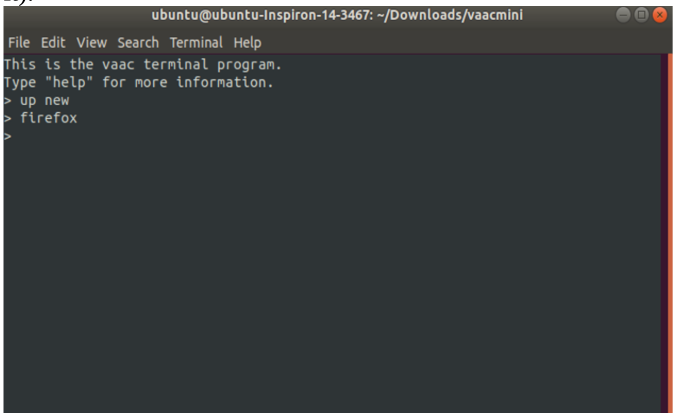
Fig. 8. Extractor Module

IV. Results And Discussions

Following are a set of results and corresponding screenshots depicting the functioning of *VAAC* -

A. Test 1: To open an application (example used: Firefox)

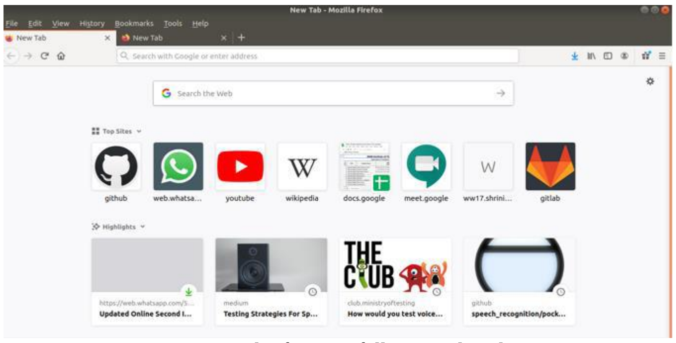
The first screenshot is the terminal view after the speech has been recognized and converted to text. In the terminal program, the converted text is analyzed to realize the target application (here, *'Firefox'*) and the action (here, to launch it).


ubuntu@ubuntu-Inspiron-14-3467: ~/Downloads/vaacmini
File Edit View Search Terminal Help

```
This is the vaac terminal program.
Type "help" for more information.
> up new
> firefox
>
```

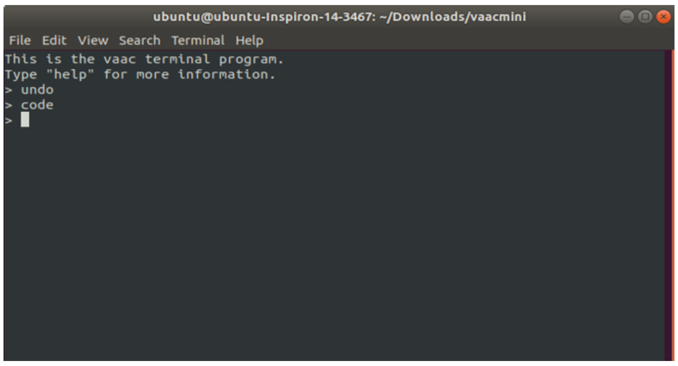
Fig. 9. Test window to open an application

The system understands clearly that the application needs to be launched and performs the action as depicted in the screenshot as seen in the next page.


New Tab - Mozilla Firefox
Search the Web
Top Sites
github
web.whatsa...
youtube
wikipedia
docs.google
meet.google
ww17.shrini...
gitlab
Highlights
Updated Online Second I...
Testing Strategies For Sp...
How would you test voice...
speech_recognition/pock...
Fig. 10. Test result of successfully opened application

B. Test 2: Ignoring unwanted commands and performing no action for the same (example used: VS Code)

Here, we command *VAAC* to '*open*' the *VS Code*. *VAAC* unfortunately for the first time does not identify the word '*open*' and instead recognizes '*undo*' (a similar sounding word). Although when spoken the second time, it recognizes and opens *VS Code*. Notably, despite of recognizing the word '*undo*' in place of code, *VAAC* does not disturb the regular flow by performing unexpected actions. Since there is no active application initially when *VAAC* is run, '*undo*' does not make any sense, it is ignored by *VAAC*.


```
ubuntu@ubuntu-Inspiron-14-3467: ~/Downloads/vaacmini
This is the vaac terminal program.
Type "help" for more information.
> undo
> code
> 
```
Fig. 11. Test window where unwanted command is uttered

VS Code is opened on the second utterance as depicted in the screenshot below.

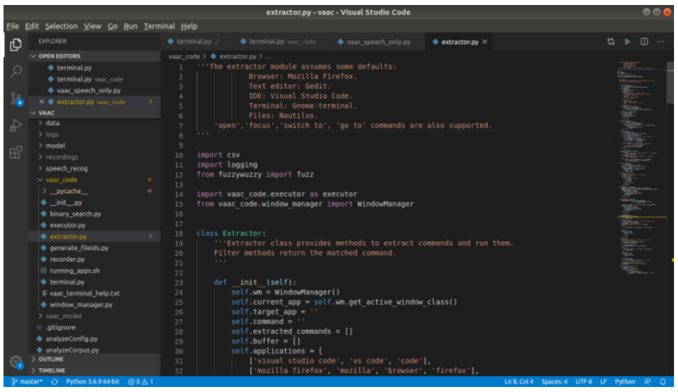
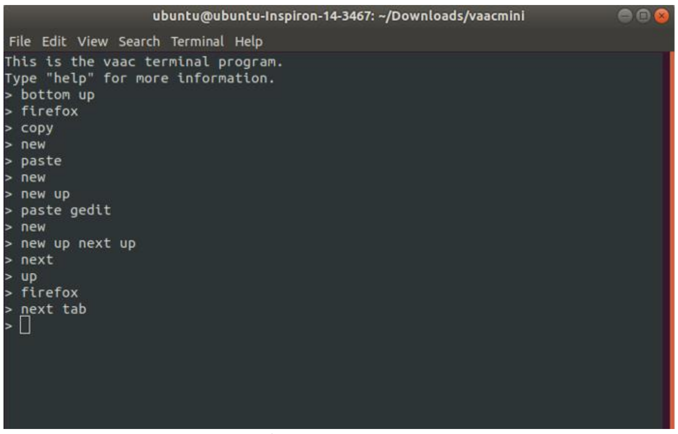

```
'''The extractor module assumes some defaults:
            Browser: Mozilla Firefox.
            Text editor: Gedit.
            IDE: Visual Studio Code.
            Terminal: Gnome-terminal.
            Files: Nautilus.
    'open','focus','switch to', 'go to' commands are also supported.
'''

import csv
import logging
from fuzzywuzzy import fuzz

import vaac_code.executor as executor
from vaac_code.window_manager import WindowManager


class Extractor:
    '''Extractor class provides methods to extract commands and run them.
    Filter methods return the matched command.
    '''

    def __init__(self):
        self.wm = WindowManager()
        self.current_app = self.wm.get_active_window_class()
        self.target_app = ''
        self.command = ''
        self.extracted_commands = []
        self.buffer = []
        self.applications = [
            ['visual studio code', 'vs code', 'code'],
            ['mozilla firefox', 'mozilla', 'browser', 'firefox'],
```

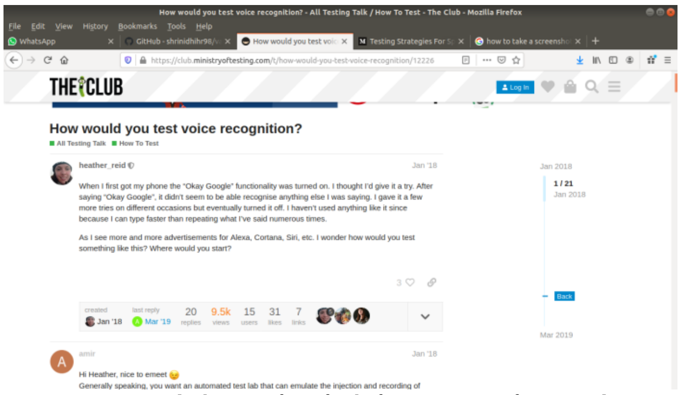
Fig. 12. Test result of VS code being opened ignoring unwanted commands

C. Test 3: Testing VAAC within an application (example used: 'Firefoz')

Initially the user commands '*Firefox*', which is executed, and the application is launched. The user has multiple browser windows open and commands *'next tab'*. *VAAC* understands the context and shifts to the next tab.


```
ubuntu@ubuntu-Inspiron-14-3467: ~/Downloads/vaacmini
This is the vaac terminal program.
Type "help" for more information.
> bottom up
> firefox
> copy
> new
> paste
> new
> new up
> paste gedit
> new
> new up next up
> next
> up
> firefox
> next tab
> 
```
Fig. 13. Test window of *VAAC* operating within an application

This is the tab that the user was browsing on initially.


How would you test voice recognition?

heather_reid Jan '18

When I first got my phone the "Okay Google" functionality was turned on. I thought I'd give it a try. After saying "Okay Google", it didn't seem to be able recognise anything else I was saying. I gave it a few more tries on different occasions but eventually turned it off. I haven't used anything like it since because I can type faster than repeating what I've said numerous times.

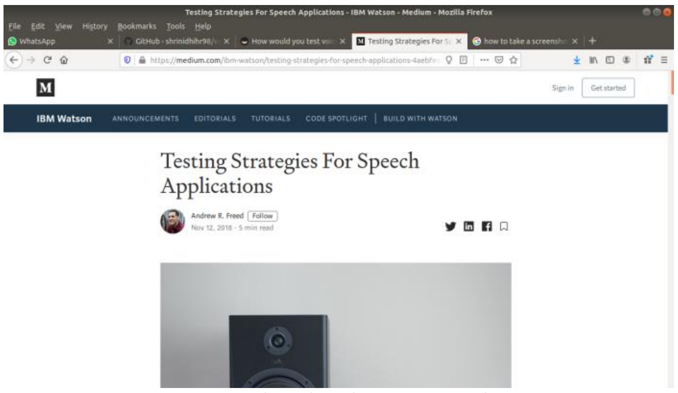
As I see more and more advertisements for Alexa, Cortana, Siri, etc. I wonder how would you test something like this? Where would you start?

amir Jan '18

Hi Heather, nice to emeet

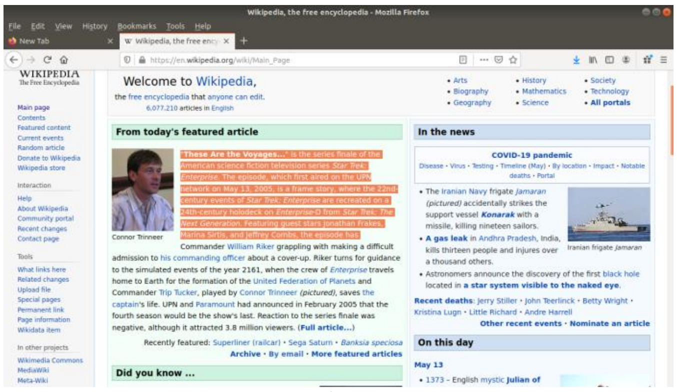
Generally speaking, you want an automated test lab that can emulate the injection and recording of
How would you test voice recognition? - All Testing Talk / How To Test - The Club - Mozilla Firefox
https://club.ministryoftesting.com/t/how-would-you-test-voice-recognition/12226
THE CLUB
All Testing Talk
How To Test
created Jan '18
last reply Mar '19
20 replies
9.5k views
15 users
31 likes
7 links
Fig. 14. Initial tab view of Firefox before execution of command

This is the screenshot after *VAAC* executes the *"next tab"* command and shifts the browser to the next tab.


Testing Strategies For Speech Applications - IBM Watson - Medium - Mozilla Firefox
https://medium.com/ibm-watson/testing-strategies-for-speech-applications-4aebf
IBM Watson
ANNOUNCEMENTS
EDITORIALS
TUTORIALS
CODE SPOTLIGHT
BUILD WITH WATSON
Testing Strategies For Speech Applications
Andrew R. Freed
Follow
Nov 12, 2018 · 5 min read
Fig. 15. Tab view of Firefox after execution of command

D. Test 4: Testing VAAC within an application (example used: 'Firefox' and gedit)

Initially, the user says '*Firefox*'. *VAAC* executes it. The user selects the text in *Firefox* and give a vocal command '*copy*'. Internally, *VAAC* copies it as shown below.



Welcome to Wikipedia,
the free encyclopedia that anyone can edit.
6,077,210 articles in English

From today's featured article

"These Are the Voyages..." is the series finale of the American science fiction television series *Star Trek: Enterprise*. The episode, which first aired on the UPN network on May 13, 2005, is a frame story, where the 22nd-century events of *Star Trek: Enterprise* are recreated on a 24th-century holodeck on *Enterprise-D* from *Star Trek: The Next Generation*. Featuring guest stars Jonathan Frakes, Marina Sirtis, and Jeffrey Combs, the episode has Commander William Riker grappling with making a difficult admission to his commanding officer about a cover-up. Riker turns for guidance to the simulated events of the year 2161, when the crew of *Enterprise* travels home to Earth for the formation of the United Federation of Planets and Commander Trip Tucker, played by Connor Trinneer *(pictured)*, saves the captain's life. UPN and Paramount had announced in February 2005 that the fourth season would be the show's last. Reaction to the series finale was negative, although it attracted 3.8 million viewers. (**Full article...**)

Recently featured: Superliner (railcar) • Sega Saturn • *Banksia speciosa*
Archive • By email • More featured articles

Did you know ...

In the news

COVID-19 pandemic
Disease • Virus • Testing • Timeline (May) • By location • Impact • Notable deaths • Portal

- The Iranian Navy frigate *Jamaran* *(pictured)* accidentally strikes the support vessel ***Konarak*** with a missile, killing nineteen sailors.
- **A gas leak** in Andhra Pradesh, India, kills thirteen people and injures over a thousand others.
- Astronomers announce the discovery of the first black hole located in **a star system visible to the naked eye**.

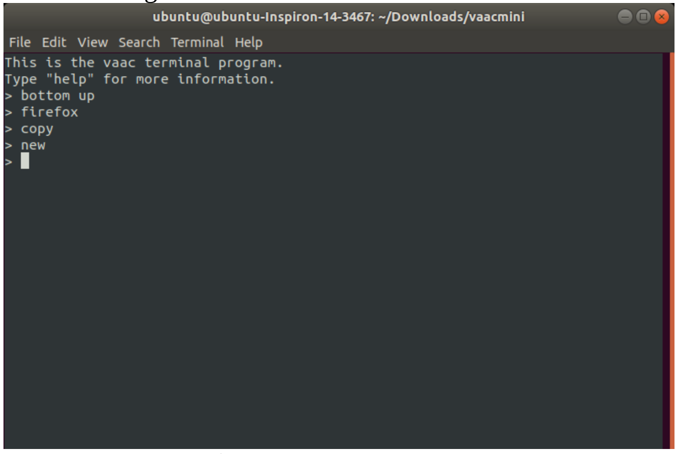
Recent deaths: Jerry Stiller • John Teerlinck • Betty Wright • Kristina Lugn • Little Richard • Andre Harrell
Other recent events • Nominate an article

On this day

May 13
- 1373 – English mystic **Julian of**

Fig. 16. Internal view of image copied by VAAC

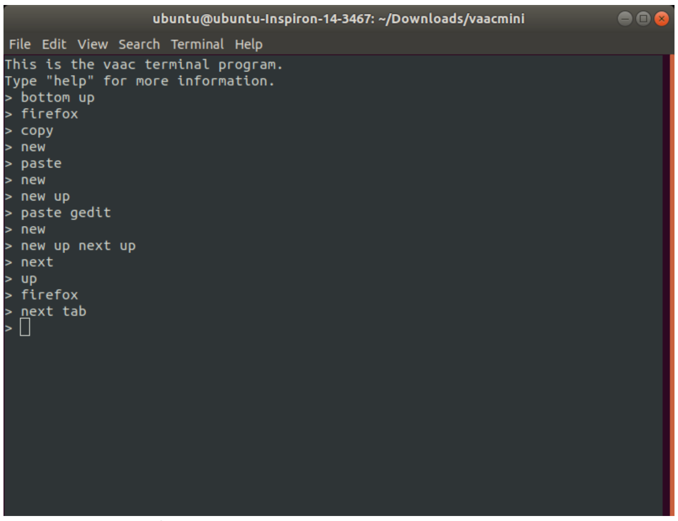
VAAC copies the selected text and ignores the word ‘*new’* which is not relevant in the current context of the application that is running.


ubuntu@ubuntu-Inspiron-14-3467: ~/Downloads/vaacmini
File Edit View Search Terminal Help

```
This is the vaac terminal program.
Type "help" for more information.
> bottom up
> firefox
> copy
> new
>
```

Fig. 17. Window view of *VAAC* ignoring commands that are irrelevant to the context

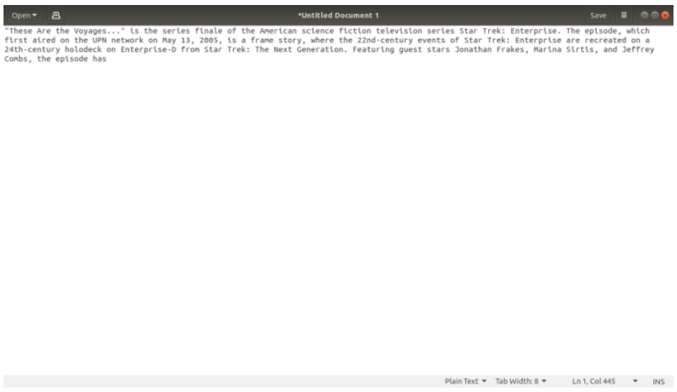
After ignoring ‘*new*’, *VAAC* recognizes ‘*paste’* but it still does not get the right target application from the user. Although it comes across unnecessary commands, *VAAC* still stores the contents and until it recognizes the command *‘paste gedit’* (*gedit* is the target application) as depicted in the screenshot below.


ubuntu@ubuntu-Inspiron-14-3467: ~/Downloads/vaacmini
File Edit View Search Terminal Help

```
This is the vaac terminal program.
Type "help" for more information.
> bottom up
> firefox
> copy
> new
> paste
> new
> new up
> paste gedit
> new
> new up next up
> next
> up
> firefox
> next tab
>
```

Fig. 18. View of VAAC recognizing commands without mentioned target

On recognizing *‘paste gedit’* command, it opens *gedit* and pastes the copied text there as shown in the screenshot below.



```
"These Are the Voyages..." is the series finale of the American science fiction television series Star Trek: Enterprise. The episode, which
first aired on the UPN network on May 13, 2005, is a frame story, where the 22nd-century events of Star Trek: Enterprise are recreated on a
24th-century holodeck on Enterprise-D from Star Trek: The Next Generation. Featuring guest stars Jonathan Frakes, Marina Sirtis, and Jeffrey
Combs, the episode has
```

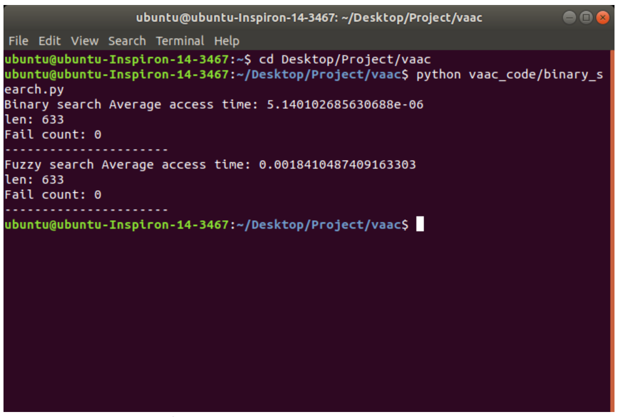
Fig. 19. Successful execution of command when target application is mentioned

V. PERFORMANCE EVALUATION

Fuzzy matching is a technique used in computer-assisted translation as a special case of record linkage. It works with matches that may be less than 100% perfect when finding correspondences between segments of a text and entries in a database of previous translations. FuzzyWuzzy is a library of Python which is used for string matching. Fuzzy string matching is the process of finding strings that match a given pattern. It uses Levenshtein Distance to calculate the differences between sequences.

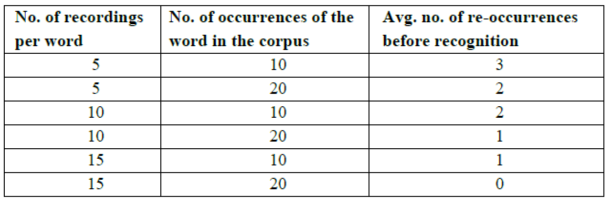
Binary search is another algorithm, also known as a half-interval search, is used for computer science to locate a specified value (key) within an array. For the search to be binary, the array must be sorted in either ascending or descending order. Binary search is much faster when compared to Fuzzy algorithm and hence contributes to the improvement in performance of VAAC during runtime where the command needs to be searched from the command lists and various filters applied once the raw text is obtained from the terminal.

As evident from the screenshot below, for a constant number of commands, the binary search turns out to be a thousand times faster than the Fuzzy search. Notably, both are applied on a sorted data since Binary search can be implemented on a sorted data only. This is another reason why Binary search is faster in this scenario.


ubuntu@ubuntu-Inspiron-14-3467: ~/Desktop/Project/vaac
File Edit View Search Terminal Help

```
ubuntu@ubuntu-Inspiron-14-3467:~$ cd Desktop/Project/vaac
ubuntu@ubuntu-Inspiron-14-3467:~/Desktop/Project/vaac$ python vaac_code/binary_s
earch.py
Binary search Average access time: 5.140102685630688e-06
len: 633
Fail count: 0
----------------------
Fuzzy search Average access time: 0.0018410487409163303
len: 633
Fail count: 0
----------------------
ubuntu@ubuntu-Inspiron-14-3467:~/Desktop/Project/vaac$
```

Fig. 20. Comparison of access time between Binary Search and Fuzzy Search



No. of recordings per word	No. of occurrences of the word in the corpus	Avg. no. of re-occurrences before recognition
5	10	3
5	20	2
10	10	2
10	20	1
15	10	1
15	20	0

TABLE I: EVALUATION OF COMMAND RECOGNITION AGAINST THE NUMBER OF RECORDINGS

VI. CONCLUSION

The core intent of VAAC was to develop a Voice User Interface-based approach for a command & control model that accepts user input in the form of speech and translates it to an executable kernel level instruction to provide improved convenience to the user in terms of device usage, across various applications. In the course of development of the project we have gained valuable insights into the software development cycle, concepts of speech analysis, entity mapping and some concepts of natural language processing.

REFERENCES

[1] W Walker, P Lamere, P Kwok, B Raj,R Singh, E Gouvea, P Wolf, J Woelfel, "*Sphinx-4: A Flexible Open Source Framework for Speech Recognition*" - CMU Sphinx-4 Speech Recognition System, Technical Report, 2004.

[2] J R. Evans, W A. Tjoland and L G. Allred, "*Achieving a Hands-Free Computer Interface using Voice Recognition and Speech Synthesis*" – IEEE AES Systems Magazine, 2000.

[3] P Warden, *"Speech Commands: A Dataset for Limited-Vocabulary Speech Recognition"* - ArXiv, Cornell University, 2018

[4] S Amann, S Proksch ,S Nadi, *"FeedBaG: An Interaction Tracker for Visual Studio"* - IEEE 24th International Conference on Program Comprehension (ICPC), 2016.

[5] T Bulmer, L Montgomery, D Damian, "*Predicting Developers' IDE Commands with Machine Learning"* - MSR 2018 Association for Computing Machinery, 2018.

[6] H Lee, Y Peirsman, A Chang, N Chambers, M Surdeanu, D Jurafsky, "*Stanford* University,Multi-*Pass Sieve Coreference Resolution System*", Stanford University CoNLL Shared Task, 2011.

[7] K Lunuwilage, S Abeysekara, L Witharama, S Mendis and S Thelijjagoda, *"Web Based Programming Tool with Speech Recognition for Visually Impaired Users"*- 11th International Conference on Software, Knowledge, Information Management and Applications, 2011