Customer Feedback

Voice Controlled Keyboard Emulator

Wirebach, Van (CS/IT) (Team Leader),
Pow Chon Long, Andrew (IT),
Schaefer, Michael (CS)

Professor Kenney (Customer)
kenney@usna.edu
CS Department Faculty (Professor)

Team Number 5, Voice Controlled Keyboard Emulator

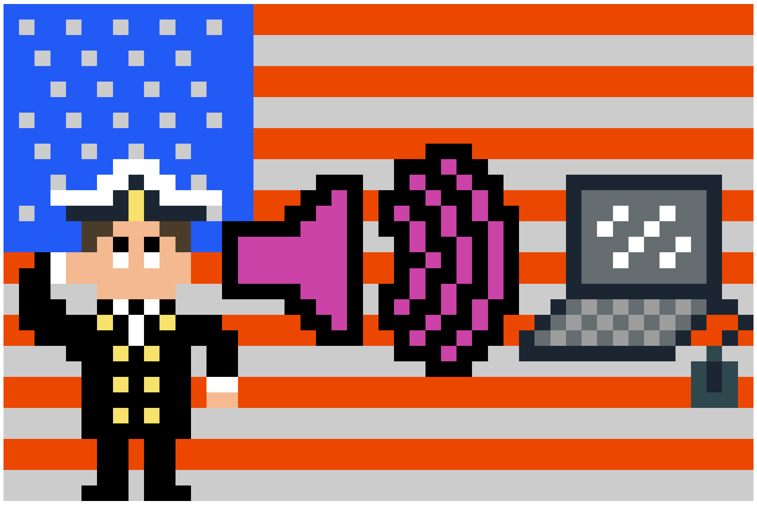


Table of Contents

System Introduction:

This manual serves to inform new users how and why to use our voice controlled keyboard emulator. By reading this, users should be able to download and install all necessary software to run our system, and know how our system works so they can start working with it.

Our system is a voice controlled keyboard emulator. By running our system, which is a single python script, in the background of your computer, you are able to type wherever your computer's mouse is currently located, by simply speaking. Our system has a simple interface to provide a few different options for how the emulator will type keys depending on the needs of your task.

Motivation:

The original motivation of this project was to provide an alternative means of writing, searching, and using the computer for disabled persons. But, as we delved deeper, we found that this could be used as a tool for all people in assisting searches, writing documents, playing games, or making notes. Therefore, we worked on making this project as accessible and universally applicable as possible.

Project Overview

Our project is to make an interface that converts voice to keyboard signals. We will demo this interface with a turn-based video game that uses keyboard input. The game will likely be programmed using Pyglet. The project will involve learning more about game accessibility, how keyboards send data across the USB, developing a video game concept, and taking that concept through all the stages of development into a well polished product.

System Set Up:

The system is a pair of python scripts and therefore set up only requires your computer to have python 3 installed and all python modules installed that the scripts require. There is no login required, however, the script must be run with sudo privileges to access the microphone. This will depend on your computer's accounts.

To download python if you do not already have it installed: follow this link to python's homepage for downloading and installing.

First, download and extract the attached zipped file.

If the file has the .zip extension, then use the file system on the computer to extract it into a folder you wish.

Otherwise, if the file has a .gz extension, navigate to the terminal and location of the file. Then, type:

```
tar -xf <file_name>
```

Substitute <file_name> for the name of the file.

Next, download all the needed python modules.

To do so, run the provided make.sh file by typing:

```
./make.sh
```

In the makefile's file location on the terminal.

Finally, run the program on the terminal.

To begin using the system, in the terminal at the location of the VoiceKeyboard.py file, type:

```
python3 VoiceKeyboard.py
```

If access is not granted, this is due to requiring root permission from the computer. In order to gain root access, run:

```
sudo python3 VoiceKeyboard.py
```

Then type in the computer's password to run the program.

The system will immediately begin running and is ready for use.

High-Level View of System:

Our system can be split into small tasks that connect sequentially. To be a voice controlled keyboard, our python script must

1) gain access to the computer's microphone
2) Listen and run a Voice To Text (VTT) API
3) Interpret the voice's text to determine what the user wants to type and
4) Send these keyboard key signals to the computer to result in an equivalent keyboard being typed

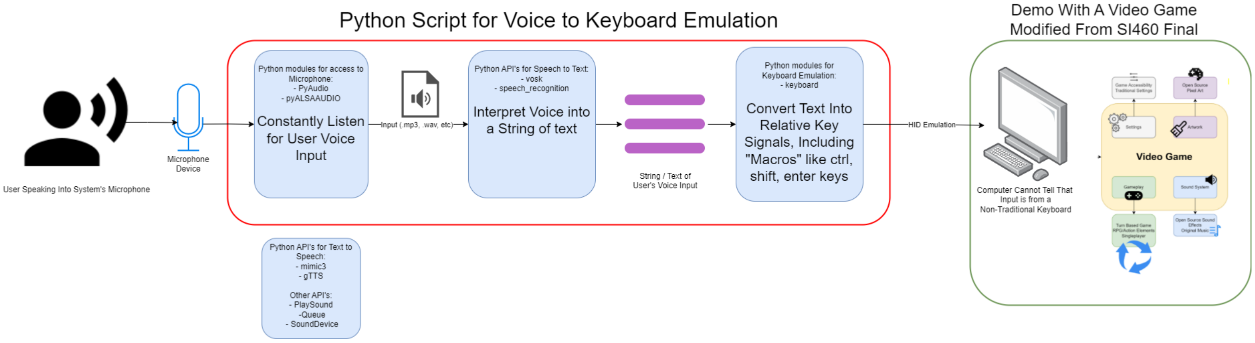
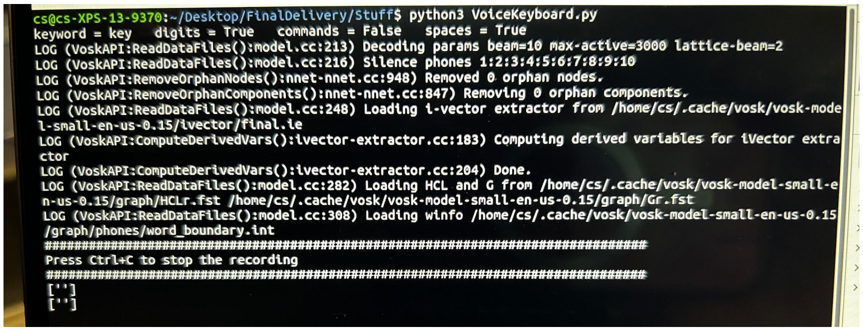


Figure 1: High Level Diagram of Our System

Quick-Start Tutorial:

Once your computer has everything installed, everytime you want to run the system simply go to a terminal, navigate to the directory the system is located in, and run:

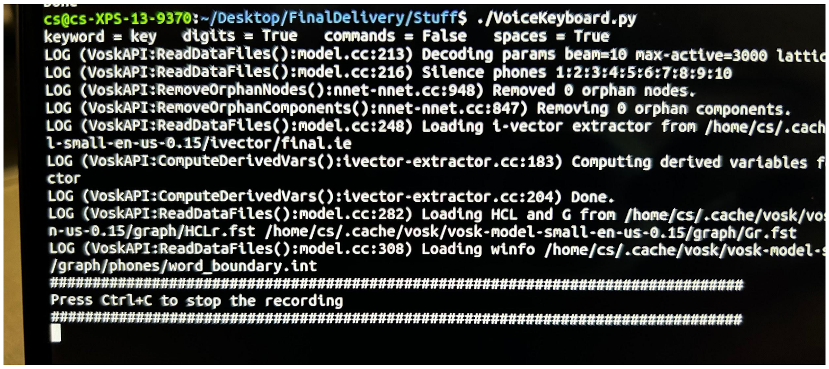
python3 VoiceKeyboard.py



```
cs@cs-XPS-13-9370:~/Desktop/FinalDelivery/Stuff$ python3 VoiceKeyboard.py
keyword = key   digits = True   commands = False   spaces = True
LOG (VoskAPI:ReadDataFiles():model.cc:213) Decoding params beam=10 max-active=3000 lattice-beam=2
LOG (VoskAPI:ReadDataFiles():model.cc:216) Silence phones 1:2:3:4:5:6:7:8:9:10
LOG (VoskAPI:RemoveOrphanNodes():nnet-nnet.cc:948) Removed 0 orphan nodes.
LOG (VoskAPI:RemoveOrphanComponents():nnet-nnet.cc:847) Removing 0 orphan components.
LOG (VoskAPI:ReadDataFiles():model.cc:248) Loading i-vector extractor from /home/cs/.cache/vosk/vosk-mode
l-small-en-us-0.15/ivector/final.ie
LOG (VoskAPI:ComputeDerivedVars():ivector-extractor.cc:183) Computing derived variables for iVector extra
ctor
LOG (VoskAPI:ComputeDerivedVars():ivector-extractor.cc:204) Done.
LOG (VoskAPI:ReadDataFiles():model.cc:282) Loading HCL and G from /home/cs/.cache/vosk/vosk-model-small-e
n-us-0.15/graph/HCLr.fst /home/cs/.cache/vosk/vosk-model-small-en-us-0.15/graph/Gr.fst
LOG (VoskAPI:ReadDataFiles():model.cc:308) Loading winfo /home/cs/.cache/vosk/vosk-model-small-en-us-0.15
/graph/phones/word_boundary.int
################################################################################
Press Ctrl+C to stop the recording
################################################################################
['']
['']
```

or

./VoiceKeyboard.py



```
cs@cs-XPS-13-9370:~/Desktop/FinalDelivery/Stuff$ ./VoiceKeyboard.py
keyword = key   digits = True   commands = False   spaces = True
LOG (VoskAPI:ReadDataFiles():model.cc:213) Decoding params beam=10 max-active=3000 lattic
LOG (VoskAPI:ReadDataFiles():model.cc:216) Silence phones 1:2:3:4:5:6:7:8:9:10
LOG (VoskAPI:RemoveOrphanNodes():nnet-nnet.cc:948) Removed 0 orphan nodes.
LOG (VoskAPI:RemoveOrphanComponents():nnet-nnet.cc:847) Removing 0 orphan components.
LOG (VoskAPI:ReadDataFiles():model.cc:248) Loading i-vector extractor from /home/cs/.cach
l-small-en-us-0.15/ivector/final.ie
LOG (VoskAPI:ComputeDerivedVars():ivector-extractor.cc:183) Computing derived variables f
ctor
LOG (VoskAPI:ComputeDerivedVars():ivector-extractor.cc:204) Done.
LOG (VoskAPI:ReadDataFiles():model.cc:282) Loading HCL and G from /home/cs/.cache/vosk/vo
n-us-0.15/graph/HCLr.fst /home/cs/.cache/vosk/vosk-model-small-en-us-0.15/graph/Gr.fst
LOG (VoskAPI:ReadDataFiles():model.cc:308) Loading winfo /home/cs/.cache/vosk/vosk-model-
/graph/phones/word_boundary.int
################################################################################
Press Ctrl+C to stop the recording
################################################################################
█
```

Tips For Use:

1) Be in a *quiet* room: The system uses VOSK, a Voice to Text translator, that picks up *everyone's* voice and always guesses when it hears something.

2) Use *short* phrases with a *pause in between* when talking to the keyboard: The system does not begin typing until it has heard your entire phrase.

3) You cannot say individual letters that can be used as a word like “b” since VOSK will interpret it as “be” or “bee.” Words that are pronounced the same way but with different meanings are ambiguous and VOSK does not understand inflection.

Modes and the Keyword:

Special Keyword: The system uses a special keyword, which can be changed before running the system, to interact with the system and help you specify how you would like the output when there is ambiguity. This is used in conjunction with the system's *modes* and the keyword is generally said *before* the thing you are trying to type / change to specify how your phrase should be interpreted.

Modes are how you interact with the system to dictate how it will *interpret* what you say.

Imagine you speak: *up down 95*
What should the voice keyboard type? The words "up" and "down" or the up and down arrows? the digits 9 and 5 or "ninety five"? Are you playing a game where the input is <up arrow>, <down arrow> and *no space* in between or are you writing a paper and want spaces in between your words? Modes answer these questions; you as the user can toggle modes on or off to suit your current needs. There are 3 modes, all of which are toggled to be True or False

1. Digits Mode: When digits mode is set to *True*, saying numbers like "one hundred" types out *10.* When digits mode is *False*, saying "one hundred" returns *one<space>hundred*

2. Commands Mode:
   a. When commands mode is set to *True*, key words like "control", "shift", and the words in the table below ("bracing", "braced", etc) default to the respective keyboard key (*ctrl, shift, [,]*). To access those actual words typed out letter by letter, say the *keyword* before the word you are speaking (if keyword is *banana:* say *banana space* to hit the spacebar)
   b. When commands mode is set to *False*, the default is not the *actual word* (saying space gets you the letters *s-p-a-c-e*). If you want the keyboard keys when commands mode is *False*, first say your *keyword* (if keyword is *key*: say *key space* to get <*spacebar*>)

3. Spaces Mode: When spaces mode is set to *True*, there will automatically be a space entered between every word you say. This would be good for a paper but might not be desired if you're typing into some input system. If spaces mode is set to *False*, there are no spaces automatically entered; spaces can still be accessed by saying space (with or without the *keyword* depending on commands mode)

To Toggle Modes: While running the system, say your <*keyword*> <*mode*> to toggle that mode. For example: if your keyword is *dog*, saying *dog spaces* will toggle spaces mode to whatever boolean value it is currently not. Remember the modes end with “s”, saying *space, digit,* and *command* will not toggle modes because the modes are: *spaces, digits, commands*

ReadFile.txt:

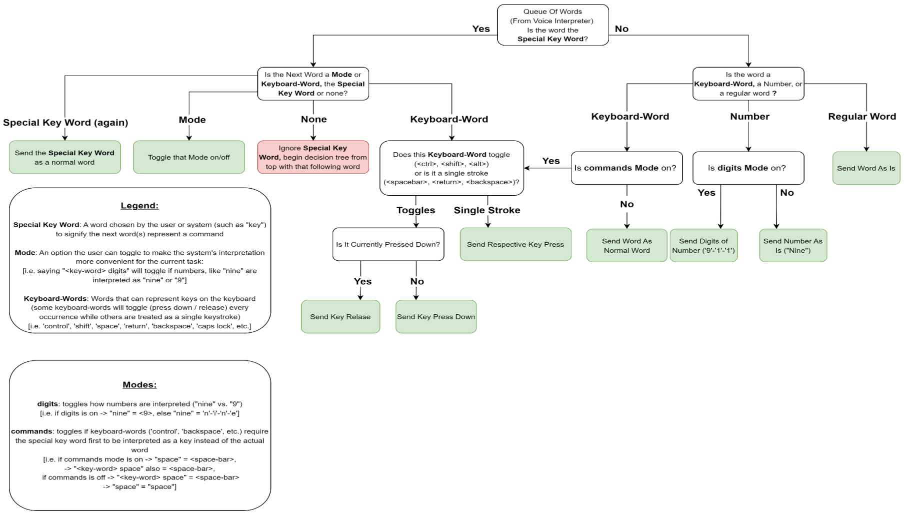
While the modes can be changed while running the system, there is a file in the directory: ReadFile.txt. This file stores the keyword and initial values of the three modes. This file can and should be edited to suit your needs. Here is an example of ReadFile.txt; do NOT change the order of these lines, only change the True vs False and desired keyword.

```
keyword banana
digits True
commands True
spaces False
```

How to Access Special Keys:

Character to Type	**What to Say to Get It**
backspace	back
+	plus
-	minus
!	exclamation
*	star
prntscrn	screenshot
.	period
volumemute	mute
capslock	caps
volumeup	louder
volumedown	quieter
nexttrack	next
|	pipe
^	carrot
%	percent
@	at
#	pound
(	hinting
)	hinted
[	bracing
]	braced
{	curling
}	curled
/	forward
\	backward
?	question

Detailed View of System:


Queue Of Words
(From Voice Interpreter)
Is the word the
Special Key Word?
Yes
No
Is the Next Word a Mode or Keyboard-Word, the Special Key Word or none?
Special Key Word (again)
Mode
None
Keyboard-Word
Send the Special Key Word as a normal word
Toggle that Mode on/off
Ignore Special Key Word, begin decision tree from top with that following word
Does this Keyboard-Word toggle (<ctrl>, <shift>, <alt>) or is it a single stroke (<spacebar>, <return>, <backspace>)?
Toggles
Single Stroke
Is It Currently Pressed Down?
Send Respective Key Press
Yes
No
Send Key Relase
Send Key Press Down
Is the word a Keyboard-Word, a Number, or a regular word ?
Keyboard-Word
Number
Regular Word
Is commands Mode on?
Yes
No
Send Word As Normal Word
Is digits Mode on?
Yes
No
Send Digits of Number ('9'-'1'-'1')
Send Number As Is ("Nine")
Send Word As Is

Legend:

Special Key Word: A word chosen by the user or system (such as "key") to signify the next word(s) represent a command

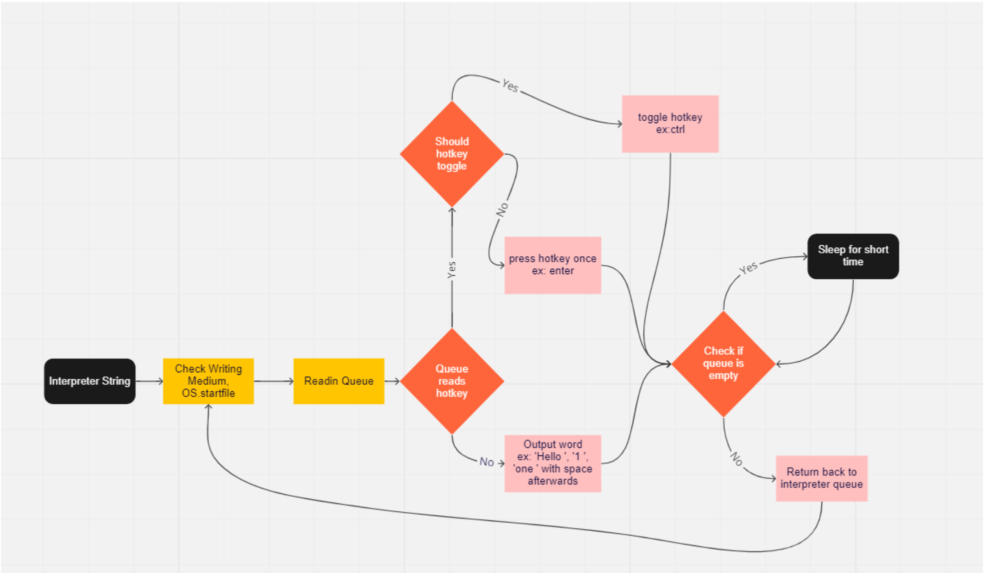
Mode: An option the user can toggle to make the system's interpretation more convenient for the current task:
[i.e. saying "<key-word> digits" will toggle if numbers, like "nine" are interpreted as "nine" or "9"]

Keyboard-Words: Words that can represent keys on the keyboard (some keyboard-words will toggle (press down / release) every occurrence while others are treated as a single keystroke)
[i.e. 'control', 'shift', 'space', 'return', 'backspace', 'caps lock', etc.]

Modes:

digits: toggles how numbers are interpreted ("nine" vs. "9")
[i.e. if digits is on -> "nine" = <9>, else "nine" = 'n'-'i'-'n'-'e']

commands: toggles if keyboard-words ('control', 'backspace', etc.) require the special key word first to be interpreted as a key instead of the actual word
[i.e. if commands mode is on -> "space" = <space-bar>,
-> "<key-word> space" also = <space-bar>,
if commands is off -> "<key-word> space" = <space-bar>
-> "space" = "space"]


Interpreter String
Check Writing Medium, OS.startfile
Readin Queue
Queue reads hotkey
Yes
No
Should hotkey toggle
Yes
No
toggle hotkey ex:ctrl
press hotkey once ex: enter
Output word ex: 'Hello ', '1 ', 'one ' with space afterwards
Check if queue is empty
Yes
No
Sleep for short time
Return back to interpreter queue

Functional Requirements Table:

Functional Requirement	Use Cases	Acceptance Test Plan Test Cases	Build #	Effort Value
1. Microphone Access: The System shall have access to the computer's microphone and be constantly listening Pow Chon Long	UC1: Hardware Access	1.1 User turns the system on / begins process. **Expected Result** -> System begins gaining access to computer microphone and starts listening. (normal) 1.2 User turns on system with all audio input devices disconnected. **Expected Result** -> System displays warning message such as "Warning, no audio input device detected". (exception)	1	2
2. Voice (Speech) to Raw Text: The system shall take an audio source as input and interpret a voice to return the string of the message spoken. Pow Chon Long	UC2: Speech-Text (VTT)	2.1 User shall speak a phrase: "Move Forward." **Expected Result ->** Interface returns string "move forward." (normal) 2.2 User shall speak muffled, incoherent sounds : "ah weortdlasdf." **Expected Result** -> System returns a warning message such as "Warning, confidence in interpretation too low to give result." (exception)	1	2

3. Raw Text to Uniform Text (digits): The system shall be able to interpret numbers in the format from the VTT interface (“one hundred”) and return interpreted string of text based on digits mode Wirebach	UC3: Raw Text - Interpreted Text	3.1 User passes system a string “one hundred” **Expected Result** -> If digits mode is on, the system will send the signals for a keyboard typing ‘1’-’0’-’0’. If digits mode is off, the system will return the signals for a keyboard typing “one hundred” like two normal words. (normal) 3.2 System receives a string with non digits in the middle of a number: “twenty x five” **Expected Result** -> System interprets the numbers as two separate numbers and the ‘x’ in the middle as a single character. (exception)	2	2
4. Text to Keyboard (Keyboard-Words): The system shall be able to interpret words that represent special keys on the keyboard like ‘shift’, ‘ctrl’, ‘return’, ‘backspace’, ‘alt’, etc. as either the word or the keyboard key depending on use of the keyword Wirebach	UC3: Raw Text - Interpreted Text	4.1 User shall speak a phrase: “<keyword> control” **Expected Result ->** Interface sends signal for keyboard toggling <ctrl> key (normal) 4.2 User shall speak the desired keyword and no special key afterwards, : “<keyword> hello” **Expected Result** -> System ignores keyword and passes result without keyword to the Keyboard (exception)	3	3

5. Text to Keyboard (Modes): The system shall maintain modes of interpretation that the user can change by speaking <keyword> <mode> Wirebach	UC3: Raw Text - Interpreted Text	5.1 System shall receive a string “<keyword> digits” **Expected Result** -> Interface will toggle digits mode and return message stating how spoken numbers will be interpreted i.e. “one” = “one” or “one” = “1”, then carry out that mode. 5.2 System shall receive a string “<keyword> commands” **Expected Result ->** Interface will toggle commands mode so keyboard words (‘alt’,’ctrl’,etc) will variably depend on <keyword> preceding it (i.e. for commands mode on “control” = “ctrl-key”) 5.3 System shall receive the mode without keyword **Expected Result** -> System ignores mode switch and returns the name of the mode as keys to the Keyboard” (exception)	4	3
6. Interpreted Text To Keyboard Signals: System shall receive interpreted text from interpreter and send keyboard signals following HID protocol for respective character	UC4: Interpreted Text - Keyboard Signals	6.1 System shall receive a string of ASCII characters “abcd” **Expected Result** ->System will send series of bytes that a connected computer interpet as the pressing down of the ‘a’ key, ‘b’ key, ‘c’ key, ‘d’ key 6.2 System shall receive string surrounded by ‘<’ and ‘>’ like ‘<control>’ **Expected Result ->** system will send the bytes representing signal of pressing	4	3

Schaefer		the CTRL key 6.3 System shall receive a string inside ‘<’ ‘>’ that has not been accounted for like <volume up> **Expected Result** -> System sends an error message “warning, not yet implemented”		

Index:

Appendices:

System Requirements: Python3, Vosk, Pyautogui, WordstoNums, NumstoWords
Installation: Pg 5
Developer Information:

Van Wirebach - m237008@usna.edu
Andrew Powchonlong - m235010@usna.edu
Michael Schaefer - m235604@usna.edu