– Independent Work 2022 –

# Analyzing Experiences with Speech Controlled Accessibility Software and Developing a Solution for the Linux Desktop

Colton Loftus
Advisor: Dr. Janet Vertesi

## Abstract

*Users on Linux often have less resources for hands-free accessibility software. In this paper I describe the design and implementation of my general purpose voice control program specifically for Linux computers. This program seeks to be both easy to customize yet general and robust in functionality. I then describe a series of user studies I carried out after having built this program. These studies were intended to better understand the reception of the program and how to better design future voice control programs. I end the paper with a series of policy and design implications as well as potential for future research in alternative computer input methods.*

## 1. Introduction

### 1.1. Motivations for Development

For individuals with disabilities affecting the use of their hands, using a keyboard or mouse can be not only inconvenient but also painful. While Windows and MacOS both have options for proprietary voice control to reduce dependency upon the hands, Linux users do not have access to these same proprietary solutions. As such, disabled individuals can be at a disadvantage or unknowingly excluded from open source software communities. By improving voice controlled accessibility software not only will disabled users benefit, but also the many programmers who are affected by pain from repetitive strain injury.

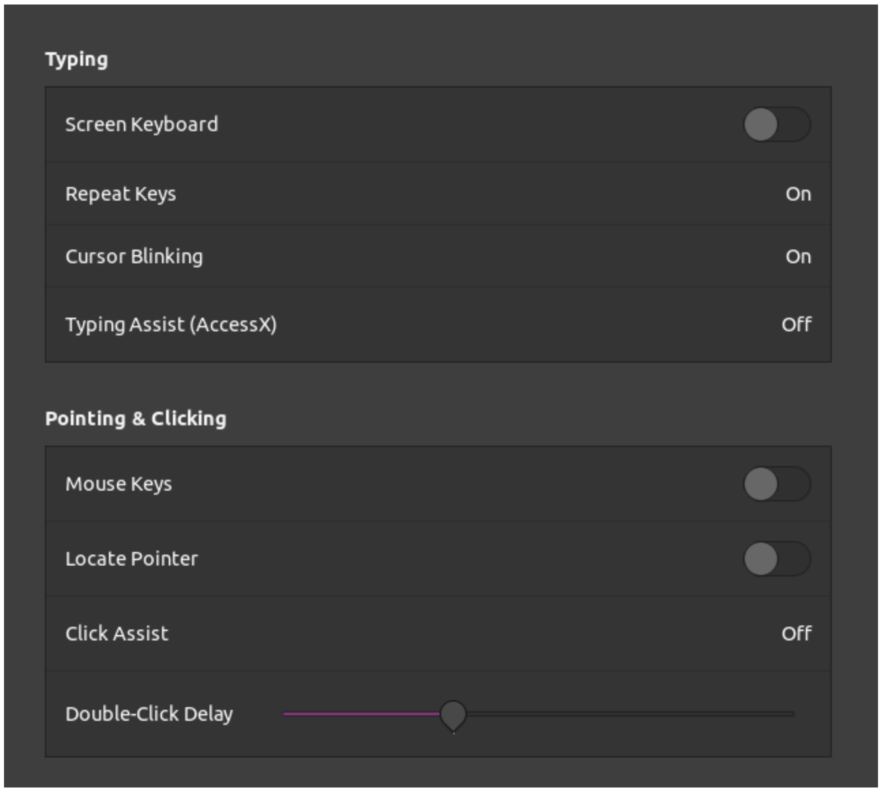
Up until this point, most research related to hands free computer accessibility is difficult to apply to general purpose voice control for the Linux desktop. There has been research applying voice control to groups with low tech literacy or intellectual disabilities with great success. [1] [4] There additionally has been research applying voice control to specific applications like social media, hardware peripherals, or games. [7] [5] However, most of this existing literature struggles to be applied to Linux users who are often technically advanced and looking for general purpose solutions for their entire desktop, not just a specific application. Much of this literature also contextualizes disability as a permanent condition. In reality, users with chronic pain syndromes may occasionally want to use different parts of voice control functionality depending on their pain level. New solutions should be found which allow for general purpose customizable voice control that can naturally interact with any type of workflow.

As a result, I wanted to not only develop a general purpose voice control solution for the Linux desktop, but also conduct a study regarding how the users interact with my software. In doing so, I hoped to gain a better understanding of how others interact with my software. This helped me to motivate my disability policy conclusions and recommendations for future designers.

## 1.2. Solutions on Linux and their Shortcomings

Before beginning to discuss the model which I will be using and the techniques for implementing it in a disability accommodation program, I first will outline the state of disability accommodation software on the Linux desktop. Currently, the most popular distribution of Linux for the desktop is Ubuntu. However, as you can see in Figure 1, it does not come with any voice control options after installation. This is the same situation in other distributions as well. While it does have accessibility features, all of them still require some use of physical keyboard presses or mouse clicks. For an individual without complete control over their hands, this can be an extreme impediment to trying out Linux and participating in software communities relating to it. This is even a problem for individuals without disabilities, like those suffering from repetitive strain injuries in the hand or wrist.

If a Linux user wishes to install additional software to fill this void, they have two main options for external voice control programs: application-specific and general purpose voice control programs. First, I will discuss the application-specific voice control options.


Typing

Screen Keyboard

Repeat Keys On

Cursor Blinking On

Typing Assist (AccessX) Off

Pointing & Clicking

Mouse Keys

Locate Pointer

Click Assist Off

Double-Click Delay

**Figure 1: The current input accessibility options on Ubuntu. Notice the lack of voice control options.**

### 1.3. Application Specific Programs

One application specific program which is particularly useful is Lip Surf, an extension for voice navigation in Chrome. [1] This application allows the user to click on links, control the scroll wheel, and input text all through voice. It can take advantage of browser accessibility tags to allow the user to have more precise link navigation. Lip Surf is a good program, but it is isolated to Chrome and has no control over any other applications or the desktop. As a result, it does not provide enough general functionality for more advanced users looking for computer-wide voice control.

Another general purpose application is Serenade. [2] This program allows for audio coding in popular programming languages like JavaScript, Python and C++. While this extension is interesting, there are a few major issues. First, the program is not meant for general purpose desktop control, even though it does have a Chrome extension. As a result, this solution does not work well

[1] https://www.lipsurf.com/
[2] https://serenade.ai/

when you have a complicated workflow. If you have both Lip Surf and Serenade open at the same time, they will both detect voice and there is no good way of synchronizing them. Serenade also has difficulty generalizing its voice commands to more niche languages and less popular development environments.

## 1.4. General Purpose Programs

The second main option for Linux is Talon. [3] Talon is a general purpose voice control engine for which individuals can write scripts to customize its behavior. Upon downloading it, Talon provides no desktop control functionality and is only a voice parser with programming APIs. However, there is a large community repository of Talon scripts called Knausj Talon. [4] These can be imported and customized as desired.

While Talon is a great piece of software, I wished to take an alternative design philosophy. First of all, in Talon adding new behavior is done through Python scripts, or its own special scripting language: .talon files. .talon files are very simple and have no support for many programming concepts like conditional control flow or calling other functions defined in a .talon file. These files are primarily sequences of commands.

While a power-user could use many Python scripts to customize Talon, it can be intimidating for individuals with limited computer knowledge. Even for those with computer knowledge, many disabled individuals might not feel that the time necessary to set it up would be worth the pain they might incur by using their hands.

I believe that using a general purpose voice control program with a simpler config system and more emphasis on scripting through interacting with the shell (instead of Python files or a new scripting language) allows for easier configuration and more flexibility. Many Linux users already have a large set of bash/fish/zsh functions which are easier to write and arguably more powerful.

Finally, one of the largest drawbacks of Talon is that the engine itself is closed-source software. While there is no reason to believe there is anything malicious done with the software, it can

[3]https://talonvoice.com/
[4]https://github.com/knausj85/knausj_talon

be uncomfortable for individuals, especially with sensitive health issues, to use a speech-to-text program that includes code that cannot be openly audited.

As a result, I thought there was a need for open-source, easy to use, and general purpose voice control programs on Linux. The application specific programs did not give the user enough control over the entire system. However, Talon seemed difficult to set up for a new user and potentially uncomfortable to privacy-conscious individuals.

## 2. Project Goals

While developing my program, I wanted to produce software that fulfilled the following goals and justifications:

- Generalizable behavior that can control the entire system
  - Generalizable behavior across many applications is often easier than having to switch between multiple application specific programs.
- A clear and verifiable policy of no data collection
  - Many software developers need to use verifiably private software for their jobs. Additionally, Linux users in general often look for and value open source private software.
- Commands as both key presses or natural language
  - In many cases, it is most natural for user to say the word which corresponds to the action that they want to perform. This is easier compared to a complicated programming command. Yet at the same time, users will sometimes want to control their computers using keyboard shortcuts they are already familiar with. Providing this flexibility pushes back on the idea that those with hand impairments necessarily have to be decoupled from keyboard input.
- Clear documentation for customizing
  - Many of the existing voice control solutions are held back by poor documentation. it is vital that new users do not feel overwhelmed when beginning to learn the software.
- Intuitive behavior for new users
  - For individuals without programming experience, it can seem daunting to have to

download scripts from Github or customize with a scripting language. While I still want the ability to have complex behavior in my program, I do not want it to have to be a necessary entry point. Thus, instead of using Python, I wanted to focus on customization through the shell and a plain text yaml config. I felt that yaml was an easy to use standard that was a natural fit for writing lists of commands.

- Full support on Linux
  - Targeting Linux especially helps me to design around the needs and demographics of Linux users. This is better than trying to target multiple operating systems and have to broaden my scope.
- Free and open source licensing
  - Most voice control solutions are not open source. Not only do I want users to be able to audit the code for privacy reasons, but I also want my program to be able to serve as a helpful starting point for future researchers. This will allow future voice control programs to be easily forked.

In summary, I above all wanted to try and create a program that made voice control feel like a natural extension of typical human computer interaction. While there are many interesting user repositories like Parrot that explore using noises like pops or clicks [5], I wanted to focus on natural speech. This design philosophy takes inspiration from Mozilla's research regarding controlling the browser with natural voice commands. [14] This research examined voice-based browser control in a way that was accessible and useful regardless of technical background. I felt like this design philosophy could be extrapolated to general purpose desktop control as well. This paper was also unique in the fact that it tried to apply voice control in a natural way that could be interwoven with other forms of computer interaction. Thus the user can interact with the keyboard if they desire, or simply use entirely voice.

All of these design goals helped me to create a program general enough to be used with the majority of software. For instance, any program with keyboard shortcuts can be controlled in

[5] https://github.com/chaosparrot/parrot.py

command mode by saying the keys. For those that cannot, the user can write shell scripts to send an application a desired action.

## 3. My Program and its Behavior

My Program is called Starling. Starlings are birds which can mimic human voices. This is also to pay homage to the disability software which influenced me: Talon and an add-on to Talon that allows for noise control (such as pops or clicks) called Parrot.

Within my program there are 4 modes. These different modes help to modularize behavior and allow the same word to be bound to different actions in different modes. Modes also help the user to better anticipate the behavior of a given voice command. Modes allow the user to type anything they want with speech to text without needing to worry about accidentally inputting a command, or vice versa. To switch mode, simply say the name of the mode.

### 3.1. Sleep Mode

Sleep mode does nothing until the user gives a command to switch the mode. The purpose of sleep mode is to be able to quickly disable the program without needing to completely exit it. This is practically useful as starting the program and loading the voice dictation model into memory takes a few seconds.

### 3.2. Dictation Mode

Dictation mode inputs the user's commands as text. This is useful for actions like writing emails, typing documents, or writing comments in code, among other things. Dictation mode is separate from command mode so the user can type a phrase like "close window" without the program interpreting that phrase as a command ( in this case to close a window for a given application).

### 3.3. Command Mode

Command mode executes key presses, window actions, or custom natural language commands. This is separated from dictation mode so the user does not need to worry about namespace conflicts.

When in command mode, the user can press any keyboard combination by saying the names of keys. While this is just one part of command mode, this is notable as it allows disabled individuals to interact with keyboard driven programs without needing to actually use the keyboard.

Since many of the letters of the alphabet are very similar, Starling defines an alphabet in the config file. This alphabet defines single syllable words in place of letter names. This is useful since many accents can make it difficult for models to correctly decode the right letter. However, if the user disagrees with my alphabet, it can be changed in the config file to any other word.

In addition to keyboard presses, command mode also allows the user to give window commands. While many desktop environments have keyboard shortcuts for controlling windows, it is usually more natural to explicitly say the desired outcome. Thus, the user can give a command to launch , close, maximize, minimize, or focus a window. Window command syntax is as follows where a window target represents the name of a window.

```
window_actions = [start, close, maximize, minimize, focus]
window_action(window_target)
```

In addition to the name of the application as it is seen by the operating system, the user can also define aliases to app names. This is since there are many programs on Linux with unnatural sounding names. For instance, it is much quicker and more natural to say terminal instead of the terminal application called urxvt. This theme of being able to use aliases throughout the config file helps to make voice commands short and natural. Instead of having to rely upon how the OS refers to an application, the user can define it how he or she wishes.

In addition to these custom aliases, there is also an automatic alias. The built in alias ’this’ will always alias to the currently focused window. This was done to add convenience and speed up a common command.

```
#  Example  when firefox is the currently focused window
#  this command will minimize firefox
minimize this
```

**Custom Natural Language Commands**

The last main part of command mode is the ability to define custom natural language commands. If you wish to define custom commands, it can be easily done in the config file. All you need to do is give the name of the application and a list of key value pairs. This continually draws upon the theme of custom behavior without overriding familiarity. If the user wishes to simply say 'ctrl-c' to copy, they still can. However, with a custom natural language command, they can bind that keypress to the voice command 'copy.'

Within the config file, each custom natural language command is given a scope. As seen in the example below, the user defines 'Mozilla Firefox' as the application, and then gives a list of commands that can be said while this application is focused and running.

```
# This example would be placed in the config.yaml file
Mozilla Firefox:
- exe_path : "firefox"
- go foward: "alt right"
- go back: "alt left"
- next tab: "ctrl \t"
- private tab: "ctrl shift p"
- new bookmark: "ctrl d"
```

This is a similar design rationale that motivated my decision to use four different modes. Splitting up and scoping behavior allows the same word to have different interpreted meanings. This is more natural and mimics human speech. The same word will inevitably be interpreted differently based on context.

### 3.4. Shell Mode

Shell mode is the last mode which can execute shell functions. This can run any alias or shell function defined in the user's shell profile. This is a useful mode since many Linux users already have scripts written to control their computers. If they do not, shell scripting on Linux is well documented and there is a vast amount of resources on sites like Stack Overflow. Since shell mode is sourcing from a shell configuration that is already present, it helps reduce the problem of unorganized configs. Using multiple application specific voice control programs can quickly create duplicate behavior, since each voice control program has its own config file. Thus, my design helps to centralize configuration and get around this issue.

## 4. Technical Outline and Development Process

### 4.1. Speech to Text Model

My program uses the 'vosk-model-small-en-us-0.15' from Vosk as its default speech to text model. [6] Vosk models are easy to download and deploy and require no internet connection. This makes them appealing from both a privacy and a development perspective. Vosk allows for the vocabulary of its models to be changed during runtime. This allows the user to have context aware inferences. For instance, users can define commands that only work when certain applications are focused. However, at the same time, the vocabulary can be unrestricted during dictation mode when the user wants a larger vocabulary for general speech. While I originally used the Nvidia Nemo toolkit with a conformer model as my default model, I found that Vosk was easier to setup and configure. Additionally, the Vosk model I used is under 50Mb. This small size made it quicker to download and set up than most other models I investigated.

### 4.2. User Audio

To get inferences from the model I recorded user audio using a stream from ffmpeg. This stream is passed to the modeland will only return an inference if it infers it to be the end of a sentence.

[6] https://alphacephei.com/vosk/models

This non-blocking stream decoder allows for better efficiency compared to a batch decoder. This is since it will continually pass audio to the model as it is hearing it in real time. I found that the batch decoder with the Nvidia conformer model I was previously using had more latency.

## 4.3. Command Parsing

After the model decodes a command, the program needs to parse how to interpret the text. Depending on the word, it could be an application name, a window action, a normal key, a modifier key, or a user defined command. I use the application mode, the user's currently focused window as well as the list of commands defined in the config file to contextually parse text into a proper command format. This format can then be passed to another function which calls xdotool to manipulate windows, Pyautogui to press keys, or other internal code to run something else.

## 4.4. Key Python Packages and External Dependencies

Beyond Vosk, I used PyGObject [7] for app indicator support on the top panel of the Gnome/KDE desktop environments. I used PyAutoGui for emulating keypresses and hotkey support.

For external depencies, I will use xdotool, xprintidle, and osd_cat. xdotool can automatically send signals to windows in the X display server (such as focus or close them). xprintidle is used for automatically detecting usage time and suggesting breaks to users (as a health feature). osd_cat prints commands to the screen for easier viewing.

## 4.5. Technical Limitations

Currently most Linux distributions use an implementation of the X Window system for their display server. However, since X is very old, some distributions are starting to move to add Wayland as an alternative display server. The issue with this is that Pyautogui, the input automation library I use, does not currently work on Wayland. However, I don't believe this will be a significant issue for a while since most distributions still include X if they include Wayland. It will take many years to phase out X, if it ever actually happens. The main other limitation is caused by the latency of the

[7] https://pygobject.readthedocs.io/en/latest/

Vosk model. It is certainly acceptable and not too slow, but proprietary models have much better inference speeds. This may be frustrating to users which are accustomed to using faster models from Apple or Google on their phones for voice dictation. Finally, since my program is written in Python and uses a fair amount of multithreading, it isn't as efficient as a program written in C or C++. However, this seemed to be a worthwhile tradeoff given that writing in Python is much quicker. Additionally, the majority of the bottleneck comes from the model which will add similar latency regardless.

## 5. Documentation Website

After implementing all the technical features, I wanted to make sure it would be straightforward for a new user to download and understand the behavior of my program. Since my program is open source, I also wanted to create a documentation website that would be easy to generate and edit if I ever had additional contributors. To accomplish this goal I used the website generator, mdbook. This program takes a directory of markdown files and uses them too create a website especially suited for documentation. All the markdown files that were used to create the website can be found in the project Github repository in the docs directory.

https://github.com/C-Loftus/Starling/tree/master/docs

The website that I created has multiple css styles which can be switched between. This is important as some disabled users have increased sensitivity to color themes with a large amount of contrast.

Just like how I described the program behavior in this paper, I split up the website into four main categories based on each of the modes. Then within Command mode it was split into four subsections due to the increased complexity of that mode.

Finally, I hosted my website on Github pages at the provided link below.

https://c-loftus.github.io/StarlingDocs/

## 6. Accessible Software Design

When designing any human interfacing software, it is vital to ensure equitable design principles. In my software project, I took multiple proactive steps to ensure my program would be efficient and fair for a variety of populations.

**Variations in Speech:** One of the biggest issues with many basic forms of voice control is caused due to variations in speech. Studies have shown speech impediments, accents, or other variations can cause significant decreases in model performance across different populations. [13][6] This is especially an issue since it also showed that the people who are affected tend to be from communities that already have challenges with equitable computer access.

I tried to address this issue in two ways. First of all, all application names can be aliased to other application names. Not every culture and demographic will use the same word for the same action. As a simple example, instead of saying "Visual Studio Code", the user could say "editor" and have the program apply the same behavior. This allows individuals to customize the name of commands while still getting the same behavior.

The next choice I did was to allow users to change the predefined alphabet. Once again, regional accents or speech impediments can make certain key names difficult for the model. To solve this, users can change key names in the config file. The alphabet is used when pressing keys. For instance instead of saying the word "c" the user says the word "cap". This allows for greater accuracy. Providing an easy way to customize key names once again can provide better accuracy without necessarily needing to change the model itself.

**Scoping:** When using voice driven interaction, sometimes a user will want to use the same command in different applications. For instance, the user may want to be able to say "close tab" in both a code editor and a web browser, but have different behavior in both. To solve this issue I drew upon the principle of implicit scoping present in a previous voice control software study. [10] Implicit scoping allows for my program to be contextually aware of the focused application whenever parsing the user command. This allows the same command to be executed differently in different contexts.

In this previous study which I took inspiration from, it noticed the user frustration that resulted from rigid scoping. This is a method of voice parsing which requires the user to state explicitly any relevant info in the voice command. While this is useful for programs with very specific commands, I did not find it appropriate to general purpose voice control. For desktop use, I wanted to take an approach similar to Talon and prioritize short commands that could be quickly decoded.

**Visual Feedback:** Finally, I wanted to provide additional ways for the user to understand the behavior of the program. I wanted the program to inform the user of how the model was interpreting the audio input. As a result, all decoded phrases are displayed on the users screen in a high contrast format. Studies have shown that providing this extra form of visual feedback can reduce user frustration and keep them more engaged. [11]

In the context of my program, this helps to reduce the feeling that voice control is something of a black box and instead give the user a better understanding of how the program is interpreting user speech. This is important as general purpose voice control has lots of behavior and could potentially be overwhelming to a new user.

**Switching Models:** In many data sets for speech to text decoding, there is a heavy imbalance of male voices over female voices, among other inequalities. For instance, in Mozilla's Common Voice dataset, it is made up of only sixteen percent females. [8] While this issue is not the main focus of my paper, I wanted to experiment with the ability to use different audio backends for my voice control software. If the user runs the following command from the make file included in the repository it will download an additional model.

```
make dev
```

This alternative model is downloaded from the Nvidia Nemo toolkit. While this does not solve any systemic issues regarding machine learning datasets, it does give the user additional options for speech to text models that might better suit their voice. I implemented this behavior since I wanted users to have an extra choice and more freedom in choosing models. Thus while I believe Vosk to

[8]https://commonvoice.mozilla.org/en/datasets

be a better fit for my application in the vast majority of cases, if the provided Vosk model does not provide good accuracy, there is another option. This also helps the future proof my software as better models become available. Future models from Vosk or Nvidia will likely continue to fit in the same software design I currently use.

## 7. User-Experience Study

After creating my software project, I ran a software study with 6 individuals. 4 individuals were college students who had significant computer experience. The other 2 individuals were middle aged adults working in an office setting who had more basic computer literacy skills. I chose these 2 groups since both students and office workers are at a heightened risk for computer repetitive strain injury. Individuals with repetitive strain injury would be one of the groups who would benefit most from my program, in addition to those with disabilities.

The purpose of these studies were to gain a better understanding of the desires of individuals when using voice control software. In addition to technical feedback, I also wanted to gain insight into how to make voice control more natural and easy to use.

This study involved three parts: a questionnaire before, a software demo, and a questionnaire afterwards. Each study took up to 2 hours each.

### 7.1. Pre-Demo Questionnaire

Before starting the demo, I wanted to be able to contextualize the experience with how knowledgeable they were with computers in general. Beyond their technical skills, I also wanted to know if they had previously used voice controlled software. If so, I wanted to know their reasons for using it. I was curious if users are primarily driven by practicality or by health reasons. These questions helped me to contextualize how the users parsed the documentation. This methodology is based on previous studies regarding how to best interview customers in a product design setting. [9] As such, I believe that many of the same principles applied to voice controlled software. I tried to use many of the same strategies, such as:

"Rather than asking people to consider hypothetical scenarios, interviewers should ask people to tell them about their current practices: what they are currently doing, how they do it, what they are trying to accomplish, what problems they face, how they handle those problems, etc." [9] I did this by asking questions such as

- What is your history with voice controlled software?
- Have you ever had a physical ailment prevents you from completing tasks on the computer?
- If you have had these sorts of issues, how have you typically gone about solving them?

## 7.2. Software Demo

Next, I performed the demo. There were three main parts of the demo. First, I presented the user with the documentation website and asked them to read it through. I was curious to see if my strategy for creating documentation would be sufficient. Next, once they had finished parsing the documentation, I asked them to perform a common task entirely with my voice controlled software. I asked them to open Google Chrome, navigate to a shopping website and add a desired product to the cart. The purpose of this task was to show the user how windows, the keyboard, and websites could all be manipulated with voice. It was also a test to see how straightforward my documentation was in the previous step. Finally, in the last part of the demo, I asked the user to create a custom command for Google Chrome. The purpose of this step was to see if my design decision of using yaml text files was an efficient choice. I also wanted to see if the user was able to build on the previous step and notice that multiple actions could be combined together in a custom command. This would allow a multistep action to be condensed into one voice command.

During the demo I used similar methodology as I employed in the questionnaire. Namely, the study suggested to provide users with questions and let them interpret it as they intended. [9] In this way, I was able to better understand the interpretive flexibility within voice controlled software use. This is in contrast to approaches that tried to prescriptively guide the user to an intended proper use of the software.

### 7.3. Post-Demo Questionnaire

After the demo, I asked about how the experience in the demo compared to other forms of voice controlled software, if they had used any before. Additionally, I wanted to get feedback regarding not only the quality of my software but also how natural it was to use. Even if the software performed well from a technical standpoint, I wanted to know how I could improve the overall experience and make voice control feel like a natural extension instead of just an accommodation for disabilities. Thus, my goal was to better understand the experience of using voice controlled software more holistically. Finally, I asked users if there was any additional behavior that they wished was included in the program.

## 8. Evaluations

### 8.1. User-Study Conclusion

When surveying all the individuals who took part in my study, I found that none had experience with general purpose voice control software. Most were only familiar with specialized programs like dictation on the iPhone or smart devices like an Alexa or google home. This was the case even for subjects who had previous challenges with repetitive strain injuries or orthopedic issues. Users with these challenges tended to work through pain or take a break altogether.

Despite this, every single user (with varying levels of difficulty during the process) was ultimately able to complete the tasks in the demo. For instance, one user with extensive computer knowledge said that

> "It was only frustrating insofar as I wasn't good at it. I think that, given some time, I could get quite comfortable with the software and really enjoy the experience."

and that it was

> "Less natural than Siri or Alexa, but much more versatile and useful in different and impactful ways."

Throughout the demo, users never reported issues with speech recognition so long as they had an acceptable microphone close enough to their mouth. This was given the fact that Vosk restricts possible vocabulary output and we were dealing with a small amount of command mode commands. However, it is important to note that dictation mode does not have this same smaller subset for its vocabulary; thus, performance was generally better in command mode than dictation mode.

At the end of the demo, I received multiple good suggestions for how to improve my program. It seems that one of the biggest challenges is how to best design documentation. This is since a general purpose voice control program has so many potential commands that can be used. One user suggested to try and create documentation with more user interaction.

> "The documentation is verbose and helpful, though I think a quickstart guide that walked you through almost more like a tutorial would be very helpful."

My current site was mainly static. While the demo was useful for learning, users said they could have benefited from sort of quiz or game format that made learning more dynamic.

Next, some users felt the process of adding custom natural language commands to be difficult. This was particularly the case for the older office workers I interviewed.

> "I had only used basic iPhone voice dictation before this. This experience showed me how voice control could do more than I originally thought and give more general purpose behavior."
> "Editing the text files felt complicated at times since I don't know anything about coding or software parsing text files"

There was some difficulty understanding the config since the name which the user interacts with is different from the name that the OS interfaces with. For instance in Ubuntu Gnome, the hard drive monitoring application is called "Disks" in the list of software pre-installed. However, when launched from the shell it has the name "gnome-disks." While I did provide a script to get app names from the focused window, this is not as elegant a solution as one might hope. Going forward, it may be beneficial to use regular expressions to match window names. Therefore '(?i)disks' would match both 'Disks' and "gnome-disks." However, new users might find this to be confusing.

Once again, this refers back to the design challenge faced in the previous paragraph. When creating general purpose voice control it is not always clear that adding extra behavior is necessarily the optimal design. Regular expressions, for instance, might solve one problem but create more confusion. Since designing for accessibility often means having to access software elements that most users do not interact with, there's not always a simple, clear solution.

Finally, some users suggested using voice control technology in environments outside the standard desktop setting.

> "I felt that the software also has other implications with other uses. For instance when driving a car or needing to multitask in some way it could be useful to have this sort of voice control."

As such, it seems that there is significant potential for future research in designing voice controlled software specifically for use outside the home, workplace or school.

## 8.2. Technology Policy Implications

**Incentivize alternatives to mouse-centered design:**

During the demo and my own testing, I found that the easiest way to input commands for a program was typically through keyboard shortcuts. Through this way, the user can simply say the names of the keys to perform the action. However, there are many applications that have poor keyboard shortcut support. As a result, they rely heavily upon clicking and the use of a mouse. This is often the case for file managers or video editors. While some have keyboard shortcuts, they are very clearly designed around the ability to drag and drop files. Users reported feeling frustrated when they tried to apply voice control to these types of programs.

As such, I believe one of the most important policy conclusions from this study is to try and incentivize other software developers to build applications with less dependence upon the mouse.

Until more software is consciously designed with accessibility in mind, lots of disability software becomes bottle-necked by the problem of mouse-centered design.

**Publicize accessibility:**

During the demo, none of my users had ever used general voice control before. As a result, I wanted to examine ways in which general voice control can be better publicized.

As stated previously in this paper, Ubuntu Linux does not come out of the box with any sort of voice control functionality. This is understandable as it might cause a significant increase in the size of the operating system for individuals who do not need this functionality. However, it would still be beneficial for OS interface designers to publicize additional tools that could be downloaded from the accessibility menu.

As I saw in the demo, many individuals, even those with extensive programming backgrounds, often don't even realize general purpose voice control software exists. Publicizing these tools, even if they are not installed by default helps to create a broader awareness for the challenges that disabled users face. It also helps to publicize the great software solutions that many are not even aware of ( this goes for any voice control program, not just my own).

**Rethink Workspaces:**

Before starting the demo, it was sometimes a challenge to even find a place to host it. This was since we needed a quiet space for the voice recognition to have the best accuracy, yet at the same time we needed the space to be one where we could talk openly. Voice dictation can be a bother to others in quiet areas like a library, yet at the same time not have good enough accuracy in louder environments like a cafe.

Thus, when designers plan out workspaces, it is important to plan these areas for those who have to work with alternative input methods. This is something that is often ignored even in contemporary workspace and office layout planning. [2] [8]

While Costa et al. and Ianeva et al. both wrote of ways to improve team work within an office setting, nothing is discussed regarding alternative input methods that may disrupt a typical layout. Voice dictation is ill suited for a collaborative office environment, yet at the same time putting a user within their own room can be isolating.

While many have argued that remote work is the most appropriate solution for those with disabilities (and thus voice controlled software), studies like those from Das et al. have pushed back on this. Disabled users often face difficulties communicating health challenges in a remote settings [3] and for those with hand impairments, it can be exhausting to need to control the computer all day for work. Thus remote work, while beneficial in some ways, still causes people with disabilities to do "additional cognitive and emotional labor beyond their day-to-day work practices to make working from home accessible." [3]

When future designers look to redesign current workspaces, emphasis should be placed on providing modular spaces which isolate sound without isolating the individual. While conducting my demos I found that the first floor of Firestone Library had a particularly great example of this. On the first floor there is a series of six glass rooms in the center of an open workspace. While others can't hear the sound within these glass rooms, those who work inside can still see out and not feel isolated. This provided a useful space for student demos that I felt could be replicated within other open office plans.

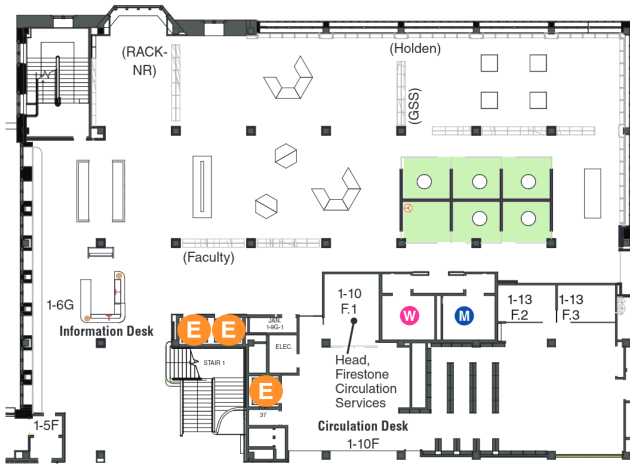


**Figure 2: Part of floor 1 of Firestone Library. Referred Workspaces in Green.**

## 8.3. Future Research to Resolve Shortcomings

While voice control is a useful option for general purpose behavior, we have seen that it also presents additional challenges to the user at times. Just like any other form of computer input, the user can develop repetitive strain if using it too much. As such, no matter how good voice control gets there is fundamentally a bottleneck due to the nature of human speech.

Voice control is also not particularly suited for actions that need to be repeated frequently in sequence. In scenarios where the user knows exactly how many times to repeat an action, they can use a macro or shell command. However, in scenarios like skimming an ebook where the user has to repeat a command many times but doesn't know exactly how many, voice control can be annoying. To be clear, there is no technical limitation here, but rather simply a feeling of annoyance caused by the bottleneck of the human voice.

Finally, depending on the context voice control is not always inappropriate input method. For instance, when speaking on a conference call, it can be embarrassing for users to have to say a voice command to mute their microphone. Additionally, in an office setting a user may want a quick way to disable voice control when a colleague comes in the room. This prevents the voice control system from picking up unintentional commands from trying to parse a random conversation. No matter how fast the voice model is, it will always be quicker to process a physical button press in these sorts of situations.

**One Potential Solution:**

While it is still in its early stages, I have explored using foot input as a way to supplement this. Dance pads, such as the type commonly used for arcade games like Dance Dance Revolution can be used as additional input methods while standing at a standing desk. When researching, I built upon previous work which used dance pad interfaces for application specific programs like a photo gallery or email. [12] I wanted to extend this research in such a way that generalized foot input across multiple different applications. Just like my goal for voice control, I wanted to make alternative input methods as customizable and robust as possible.

As a result, the user does not need to use their hands or any sort of voice. This allows for inputs which can be quick, precise, and useful in situations where the user cannot speak commands. The repository for this program can be found at the link below. This program can be used simultaneously with voice control.

https://github.com/C-Loftus/step-hotkey

As of spring 2022, this repository currently shows a proof of concept. Pressing a key on the dance pad will trigger in event specified in the config file. This can mimic a voice command, a shell command, or a keypress. Thus, it is very similar to how I designed command mode in my voice control program. While this program may seem silly upon first glance, I do think that there is legitimate potential in being able to leverage foot control will standing at a standing desk. It is very natural to simply hold down a button with your feet to scroll down. This makes it so you do not need to repeat the scroll down command multiple times with your voice. Additionally, it can help prevent repetitive strain injury by diversifying your input methods between feet, hands, and voice. Finally, there is also potential given the fact that dance pads can be bought for less than $30, compared to other forms of computer input, like eye trackers which can retail for $200.

## 9. Final Notes

In summary, I hope that I have created not only a novel and useful software program but also design research and a code repository that will provide a useful inspiration for any future designers in voice controlled software.

My project shows how designers have a series of complicated decisions whenever they work with voice controlled software. Advanced users often want customizable behavior, but too much customization can alienate beginners. Privacy conscious users often do not want to use tech companies' cloud voice recognition APIs, yet open source speech recognition alternatives require more setup. Some users want voice control simply as a more natural way to type, while others depend upon voice control due to their disability. Finally, even if all the technical challenges are met, poor workplace layouts and software design can impede usage. Thus, I hope that this paper has shown not only the technical challenges of general purpose voice controlled software, but also the many social and design challenges that come with alternative input methods.

## 10. Acknowledgments

I would like to thank all the participants in my software study for taking the time to help me in my research. I would like to thank my junior independent work advisor, Dr. Janet Vertesi at Princeton University, who helped direct me during the research process.

## References

[1] S. S. Balasuriya *et al.*, "Use of voice activated interfaces by people with intellectual disability," in *Proceedings of the 30th Australian Conference on Computer-Human Interaction*, ser. OzCHI '18. New York, NY, USA: Association for Computing Machinery, 2018, p. 102–112. Available: https://doi.org/10.1145/3292147.3292161

[2] V. G. J. Costa and C. França, "How office layouts influence software development?" in *Proceedings of the 34th Brazilian Symposium on Software Engineering*, ser. SBES '20. New York, NY, USA: Association for Computing Machinery, 2020, p. 173–182. Available: https://doi.org/10.1145/3422392.3422441

[3] M. Das *et al.*, "Towards accessible remote work: Understanding work-from-home practices of neurodivergent professionals," *Proc. ACM Hum.-Comput. Interact.*, vol. 5, no. CSCW1, apr 2021. Available: https://doi.org/10.1145/3449282

[4] K. Dubey *et al.*, "Learnings from deploying a voice-based social platform for people with disability," in *Proceedings of the 2nd ACM SIGCAS Conference on Computing and Sustainable Societies*, ser. COMPASS '19. New York, NY, USA: Association for Computing Machinery, 2019, p. 111–121. Available: https://doi.org/10.1145/3314344.3332503

[5] X. T. Gao *et al.*, "Assist disabled to control electronic devices and access computer functions by voice commands," in *Proceedings of the 1st International Convention on Rehabilitation Engineering amp; Assistive Technology: In Conjunction with 1st Tan Tock Seng Hospital Neurorehabilitation Meeting*, ser. i-CREATe '07. New York, NY, USA: Association for Computing Machinery, 2007, p. 37–42. Available: https://doi.org/10.1145/1328491.1328502

[6] M. Garnerin, S. Rossato, and L. Besacier, "Gender representation in french broadcast corpora and its impact on asr performance," in *Proceedings of the 1st International Workshop on AI for Smart TV Content Production, Access and Delivery*, ser. AI4TV '19. New York, NY, USA: Association for Computing Machinery, 2019, p. 3–9. Available: https://doi.org/10.1145/3347449.3357480

[7] C. Holloway *et al.*, "Disability interactions in digital games: From accessibility to inclusion," in *Extended Abstracts of the Annual Symposium on Computer-Human Interaction in Play Companion Extended Abstracts*, ser. CHI PLAY '19 Extended Abstracts. New York, NY, USA: Association for Computing Machinery, 2019, p. 835–839. Available: https://doi.org/10.1145/3341215.3349587

[8] M. Ianeva, P. Chotel, and F. Miriel, "Learnings from workplace user-centered design: The case of a media and communication company," in *Proceedings of the European Conference on Cognitive Ergonomics 2015*, ser. ECCE '15. New York, NY, USA: Association for Computing Machinery, 2015. Available: https://doi.org/10.1145/2788412.2788426

[9] E. A. Isaacs, "Interviewing customers: Discovering what they can't tell you," in *CHI '97 Extended Abstracts on Human Factors in Computing Systems*, ser. CHI EA '97. New York, NY, USA: Association for Computing Machinery, 1997, p. 180–181. Available: https://doi.org/10.1145/1120212.1120332

[10] F. James and J. Roelands, "Voice over workplace (vowp): Voice navigation in a complex business gui," in *Proceedings of the Fifth International ACM Conference on Assistive Technologies*, ser. Assets '02. New York, NY, USA: Association for Computing Machinery, 2002, p. 197–204. Available: https://doi.org/10.1145/638249.638285

[11] Y. Liu *et al.*, "Supporting task resumption using visual feedback," in *Proceedings of the 17th ACM Conference on Computer Supported Cooperative Work amp; Social Computing*, ser. CSCW '14. New York, NY, USA: Association for Computing Machinery, 2014, p. 767–777. Available: https://doi.org/10.1145/2531602.2531710

[12] B. Meyers *et al.*, "Dance your work away: Exploring step user interfaces," in *CHI '06 Extended Abstracts on Human Factors in Computing Systems*, ser. CHI EA '06. New York, NY, USA: Association for Computing Machinery, 2006, p. 387–392. Available: https://doi.org/10.1145/1125451.1125534

[13] E. Mubarak *et al.*, "Does gender and accent of voice matter? an interactive voice response (ivr) experiment," in *Proceedings of the 2020 International Conference on Information and Communication Technologies and Development*, ser. ICTD2020. New York, NY, USA: Association for Computing Machinery, 2020. Available: https://doi.org/10.1145/3392561.3397588

[14] A. C. Williams *et al.*, “Toward voice-assisted browsers: A preliminary study with firefox voice,” in *Proceedings of the 2nd Conference on Conversational User Interfaces*, ser. CUI ’20. New York, NY, USA: Association for Computing Machinery, 2020. Available: https://doi.org/10.1145/3405755.3406154