SpeedNav: Document Navigation By Voice

Andrew Begel, Zafrir Kariv
University of California, Berkeley
abegel@cs.berkeley.edu, karivkln@uclink4.berkeley.edu

October 9, 2002

Abstract

This paper examines document navigation using voice recognition software. Existing commercial tools' support for voice-based navigation provides a substandard replacement for those whose disabilities prevent them from using the keyboard and mouse. We show, in a GOMS analysis, that the two main limiting factors of any navigation method are the interactive speed of the recognizer, and the cognitive load of the task presented to the user. Our tool, SpeedNav, was designed to address the current situation by enabling users to navigate via an auto-scroll mechanism, reducing both the number of commands spoken (incurring less delay) and the cognitive load (enabling the user to focus more on scanning the text for the desired target than on issuing navigation commands). The results of our user study with speech recognition experts are presented within.

1 Introduction

There are many people who suffer from Repetitive Strain Injury (RSI), and others from more severe motor impairments, who cannot easily use a keyboard and a mouse. For these individuals, staying productive in a work and home environment that increasingly relies on computers for creating and editing documents is difficult.

One form of assistive technology that has the potential to remedy such disadvantages is speech recognition. Speech recognition enables one to speak into a microphone and have what one says translated into actions for the computer to perform (commands) or transcribed directly into a document (dictation). It has been around for over 40 years, yet only in the past five years has it really become usable on personal computers. There are three major commercial products that support voice recognition on the desktop: IBM ViaVoice [4], Scansoft's Dragon Naturally Speaking (and Dictation) [15], and Microsoft Office XP [13].

Speech recognition used as a replacement for the keyboard and mouse is not the solution for everyone. Recognition accuracy suffers when users have accented speech, speech impediments, inconsistent prosody (such as if the user has a cold or is hoarse), or, if used in a typically noisy environment. Numerous studies show problems with the basic usage of speech recognition, including errors (due to both the recognizer and the user, especially during misdictation correction) [8], limited human working memory capacity for speech [10], and the unforgiving nature of restrictive acceptable voice input [14]. In addition, users must re-learn basic word processing techniques using voice rather than keyboard and mouse. These problems and the steep initial learning curve cause, for most users, speech recognition-based editing to be significantly (and often prohibitively) slower than editing using keyboard and mouse.

With few exceptions ([9]), all of the research exploring deficiencies in speech recognition has concentrated on dictation, which has prompted commercial speech recognition manufacturers to im-

prove it. However, the process of document navigation has not been subject to similar consumer pressure, and as as a result, has stagnated. To this effect, current methods of document navigation are cumbersome and difficult to use.

Through our GOMS analysis of documentation navigation in Section 3.1, we show that navigation by speech is limited by two main components:

1. **Speech recognition performance** which slows down interactive use.
2. **Cognitive Load** of the task as presented to the user.

We hypothesize that reducing the number of spoken commands (which reduces the total time spent in speech recognition), and reducing the number of actions that require willful thought (which reduces the cognitive load of the task) will make it possible to create a faster and easier-to-use navigation by voice mechanism.

In this work, we have designed, implemented, and tested several ways to increase the speed and utility of speech-based document navigation methods, while reducing the cognitive load at the same time. In the following paper, we introduce the basics of speech recognition, discuss our numerous designs, influenced by interviews with experts in speech recognition, and explore the ramifications of those designs gained by looking at prototypes and trying them out for ourselves. We developed criteria for the design of widely-applicable document navigation methods. Two of our designs survived this phase, and were tested with expert users of voice recognition. We present the results of our user study which turn out to be inconclusive as to whether our methods improve on existing techniques. Finally, we conclude with several new ideas to improve on the voice-based navigation techniques developed herein.

2 Speech Recognition Basics

The primary use of speech recognition is the creation and maintenance of text documents. A user begins by dictating into a microphone, whereupon the speech recognizer translates the speech into text for later insertion into a word processor. Speech recognizers often support some automatic formatting (such as capitalizing proper nouns and the beginning of sentences), but usually require the user to explicitly verbalize punctuation and document formatting commands (such as boldface, italics, etc).

Once the user has finished dictation, they must use the voice recognizer to navigate and edit their document. All three commercial speech recognition packages support similar interfaces for these tasks. A user can say a short phrase to cause the cursor to move or to perform an editing action (cut, copy, paste, boldface, italics, etc).

Navigation commands usually involve no document content at all (e.g. move down 2 lines, go to the previous page). We call this kind of cursor movement *relative navigation.* Dragon's Naturally Speaking supports Select-n-Say, where a user is able to augment a navigation command with an explicit phrase from the document (e.g. Select the sentence that starts with 'The quick brown.') in order to speed the process. In addition, all packages support mouse grid, a means of addressing a pixel location on the screen using a hierarchical 3x3 grid that is overlaid on the screen. Mouse grid usually requires five to six commands (one to bring it up, three or four to navigate to the target, and a final one to select it). We call these kinds of cursor movement *direct navigation.*

These designs are naive and have several flaws. In relative navigation there are too many words to speak; using the keyboard is much faster (average typists can type around 5 keys per second [2]). In addition, the words require the user to estimate cursor distances to the desired screen location. If this location is more than some threshold line/characters away, the user has to guess the distance, or spend time to explicitly count the distance. If the location is off the screen, the user must jump long distances and then correct for any over or undershoot. In addition, there is no auto-repeat support in commercially available speech recognizers, as there is on a keyboard. The user must repeat the navigation phrases over and over again until the de-

sired location is found.

Select-N-Say requires the users to read and understand the text on the screen before uttering their navigation command. This incurs more cognitive load than using the keyboard or mouse to move the cursor. Assume the user reads at a rate of 260 words per minute [2], and the average search phrase is 5 words long. If the user has already spotted the phrase on the screen, it will take him a little over 1 second to read the phrase. At a dictation speed of 80 words per minute, it will take another 3 seconds to speak it, for a total of 4 seconds of activity for the task. By stark contrast, the time it takes to verbalize a location uniquely and accurately is far greater than the time needed to point to it with a mouse.

In addition, Select-N-Say only works when the location is on the screen and visible. If the words at the desired location are not unique, the user must include nearby, but unrelated words to form a unique search phrase. If the user finds that words they want to speak are difficult for the speech recognizer to recognize reliably (such as the homophones: to, two, too and 2), they must either avoid speaking them in their search, or suffer the slowdown due to arduous dictation correction facilities.

This analysis of document navigation techniques is high-level. In the next section, we use GOMS to undertake a more formal analysis.

3 Analyzing Navigation Techniques

A look at cursor navigation techniques brings to mind Fitt's law [3], which states that the time it takes for a person to point at a location in space is

$$time \propto log(\frac{2 \times distance}{target\ size})$$

This work has been extended by others [11] to the action of pointing at a target on computer screen with a mouse. In addition, later work has explored moving a mouse manually along an on-screen path to develop the Steering Law [1].

$$time \propto \frac{distance}{width\ of\ path}$$

Karimullah and Sears [9] studied cursor navigation using voice commands to move the cursor to an on-screen graphical target. Unique to their study, Karimullah and Sears enabled the users to control the cursor's velocity rather than its position.

While each of these techniques appears to approximate the document navigation task, there are important differences. Our task is multi-page; the target of navigation is usually not initially visible, and must be scrolled into view. Second, our task involves scanning for a target phrase in the midst of a page of text, not merely spotting a sole graphical target. Thus, Fitt's law and its extensions are not applicable for our task. Therefore, we see the need to develop a new GOMS model of text document navigation using speech recognition.

3.1 GOMS Analysis for Navigation

We employ the GOMS usability modeling technique for predicting how much time several common document navigation techniques will take. Applying the KLM variant of GOMS [7], we illustrate how the keyboard, mouse, and voice-activated operators involved in these techniques combine to form a complete timing measurement. The following numbers apply only to experts in both keyboard-based and voice-based navigation techniques. We are directly comparing the two in order to gain an understanding of the disadvantages afforded by the current voice-based techniques on those who cannot use keyboard and mouse.

The most important factor in the task of searching through a text document is not distance to the target (especially since the target is not often on the screen), but instead how recognizable the target phrase is. This is related to what you are looking for, what the actual words are, how unique they are, how fast you can read and comprehend the text, and especially whether or not you know the exact wording of what you're looking for or have only a vague knowledge of its contents.

In Figure 1, we describe the equations that govern the time it takes to navigate n lines in a text document to a desired target.

Equation #1 shows that the number of lines a

$$d_{lines} = \sum_{-n \le i \le n} i c_{l_i} \quad (1)$$

$$t_{scan_n} = \alpha n + \beta \sum_{-n \le i \le n} c_{l_i} + \gamma \sum_{-m \le j \le m} c_{s_j} \quad (2)$$

$$t_{nav} = (1 + \rho_{error}) \sum_{-n \le i \le n} c_{l_i} (\Delta_{rc} + t_{scan_i}) \quad (3)$$

d_{lines}	=	distance in lines to the target
Δ_{rc}	=	computer's recognition delay per command
c_{l_n}	=	number of commands to scroll n lines
c_{s_j}	=	number of commands to speed up or slow down to speed m
α, β, γ	=	multipliers of components of t_{scan_n}
t_{scan_n}	=	time to scan n lines to look for the target
ρ_{error}	=	the sum of voice recognition and user error rates
t_{nav}	=	total time to navigation from start to target

Figure 1: Supporting equations for the GOMS model.

given command scrolls (e.g. down arrow scrolls one line, page down scrolls 24 lines) multiplied by the number of times that command was given equals the number of lines traveled.

Equation #2 shows that the user's scanning time is proportional to the number of lines read and the number of scrolling and speed changing commands given (each command may take the user's mind off the scanning process)[1].

Equation #3 shows that the total navigation time is equal to the summation for all commands that move n lines times the delay in recognition plus the time the person needs to scan the text after scrolling. This summation is then multiplied by one plus the expected error rate (recognizer error and user error).

Let's look at each type of navigation and run them through the equation. Given a navigation that is d_{lines} away, and a set of commands that enables you to navigate by any number of lines or any number of screens (equivalent to some number of lines), and given a target phrase-human combination that imposes a definite effect on t_{scan_n}, we can vary some variables in the following navigation methods:

- **Keyboard navigation**: Δ_{rc} is very small, around 70 ms [2]. The error rate is close to zero. $c_{l_1} = 1$, $c_{l_{24}} = 1$ (assuming 24 lines per page), and for $2 \le n \le 23$, $c_{l_n} = n$. Auto-repeat on the keyboard, lowers successive Δ_{rc}'s to 33 ms (assuming 30 cps repeat rate).

- **Speech-based navigation by discrete jumps**: Δ_{rc} for speech recognition using IBM ViaVoice on a Thinkpad T20 P3-700/512 is 750 ms. The error rate is around 5%. c_{l_n} for any n = 1 (using the command "go down n lines").

- **Find Dialog by keyboard**: People use the find dialog only when they know the exact word(s) that they are looking for. Equation #3 is not representative of the find dialog. The following equation more closely approximates the task time.

$$t_{find} = c_{find}\Delta_{rc} + t_{typing} + c_{OK}\Delta_{rc} + (E(P_{find}) - 1)(c_{next}\Delta_{rc} + t_{scan}) \quad (4)$$

Users operate the find command by first issuing a command to open the find dialog, then typing in the words they are looking for, and hitting the OK button. For each successive search result highlighted by the system, the user must scan the line to see if the desired words were

[1] See discussion in Section 4.2

found; if not, the user issues a find-next command and repeats. The expected number of times to repeat is half of the number of times the words appear in the target document, assuming that the user's target is uniformly distributed amongst the search results.

In this mode, $\Delta_{rc} = 70$ ms, $c_{find} = 1$, $c_{OK} = 1$ and $c_{next} = 1$.

- **Find Dialog by Voice**: This is the same as Find Dialog by Keyboard, except that $\Delta_{rc} = 750$ ms, and t_{typing} is replaced by $t_{dictation}$, the time it takes to dictate (with errors and error correction) the target phrase into the find dialog. c_{find} is usually 2 (Edit menu – Find menu item), but can be 1 with a speech macro. $c_{OK} = 1$, and c_{next} is usually 2 (Edit menu – Find Next menu item), but can be 1 with another speech macro.

- **Select and Say**: This is similar to the find dialog, but the target phrase must be visible on-screen, thus we must add in the scrolling time to make the line visible on-screen to the equation. The equation is as follows:

$$t_{selectandsay} = t_{scroll} + t_{scan} + t_{dictate} \quad (5)$$

t_{scroll} is the time to scroll within one screen of the target using any of methods described above. t_{scan} is the time it takes for the person to find the desired target on the screen (related to a user's skimming capability) and is inversely related to the target's uniqueness. Once the person finds the target, they must speak it out loud ($t_{dictate}$) and then the software hilights the phrase.

As one can see, the two dominant controllable factors here are t_{scan} and c_{l_n}, the scanning time and the number of commands issued by the user. Reducing any of these numbers should result in faster navigation times.

4 Design

Our design phase started with a survey of expert voice recognition users for their impression of existing voice recognition packages, with emphasis on their use of the editing facilities. We sought to incorporate the expert's suggestions for improvement into our product.

4.1 Expert Interviews

We interviewed several experienced users of speech recognition – those who use speech recognition for most of their work during the day. Several issues stood out amongst all the experts.

1. The speed and quality of voice recognition was always described as too slow, too cumbersome to use, too unreliable, bad at recognizing accents, and error-prone. These were the biggest issues by far. When described in terms of frustration, one particularly good user said he experienced 2-3 frustrating moments per hour. Usually if a user had the use of their hands, they would "cheat" and revert to using a keyboard whenever voice recognition began to fail them.

2. Experts use voice recognition in different ways. Some use it only for dictation, and perform editing by hand. If any editing by voice was performed, it would be a short period of time after dictation (such as completing a paragraph). Many find that it does not work in all needed applications, and does not work in technical applications (e.g. for computer programming).

3. People had problems editing by voice. In particular, they could not easily verbalize where they wanted the cursor to go, nor could they easily figure out how to command the cursor to go there. For some, using the mouse grid feature was the only reliable way they could move the cursor to a location they could see on the screen. All those we interviewed recognized

that speech macros could be useful to speed up this task, but most did not use them.

4.2 Prototype Ideas

We developed many ideas towards achieving our goals. We prototyped six of them in Microsoft Word using Visual Basic. These prototypes were useful in furthering our understanding of the difficulties in designing a good navigation tool, and we present our findings here.

Our major idea was auto-scrolling with a voice-enabled speed control. To reduce the number of commands the user must say, the user can start scrolling the document in a particular direction, scanning the document as it scrolls by, looking for the desired phrase, and then stop it when they see their target on the screen. We assert that this also reduces the cognitive load to merely scanning the text as it scrolls by – the user does not have to re-utter the text in order to select it. We also add a speed control, enabling the user to match the scrolling rate to his natural scanning capability.

To validate our idea, we extend the GOMS analysis (from Section 3.1) for the following two methods: auto-scroll and auto-scroll with speed control.

Speech-based navigation by auto-scroll: In this mechanism, there are only two commands necessary, "start" and "stop", so $c_{l_{d_{lines}}} = 2$, and for all other n, $c_{l_n} = 0$. The time between successive scroll actions is t_{pause}. This is set to a particular value (the next mechanism enables speed control) but must be greater than the t_{scan} or the person will not have enough time to comprehend the text before it scrolls away. $\Delta_{rc} = 750$ ms, so the drop in the number of commands to a constant two commands should have a significant effect to lower the overall navigation time. The velocity of the scrolling motion directly affects the precision of the user to avoid overshooting and indirectly affects accuracy of cursor placement.

Speed-based navigation by speed-controllable auto-scroll: Finally, we take the above mechanism and add two commands to control the pause time: "faster" and "slower". We hypothesize that the desired navigation speed of the user will follow the equations below:

$$t_{start} = \Delta_{rc} \quad (6)$$

$$t_{stop} = t_{scan_n} + \Delta_{rc} \quad (7)$$

$$t_{pause} \geq t_{stop} \quad (8)$$

t_{start}	=	time it takes for the system to react to a start command
t_{stop}	=	time it takes for the system to react to a stop command
t_{pause}	=	time between successive actions by the editor

Once the user begins scrolling, the system sets the pause time to the user's personal initial value. The user may issue the speed control commands to raise and lower this pause time, but may not lower the time below the time he needs to scan n lines of text for the target. The pace of scrolling will drop (pause time will increase) as the user nears the target. While the Δ_{rc} is 750 ms for the "start" and "stop" commands, we feel it is possible that the "faster" and "slower" commands overlap the pause time, therefore its contribution to t_{scan} is zero (This finding requires further study). Most users do not realize that Δ_{rc} is so long, and therefore they do not slow down in time, and overshoot the target. This overshoot problem will diminish (but never go away) with practice.

4.3 Prototype Results

We implemented six variants of the above idea. The first was to auto-scroll the cursor (not the document) one line at time, enabling the user to start, stop and control the speed of the cursor motion. We learned that human eye tends to follow moving objects (e.g. the cursor), in preference to reading the text of the scrolling document. In addition, Microsoft Word has a undesirable property of leaving the cursor at the bottom of the page when it is scrolling down, preventing the user from reading anything below it.

We noticed a far more insidious problem of *cursor overshoot*, which we believe to be inherent in any type of auto-scrolling solution. The user (mediated by the speech recognition software) has a

minimum reaction time (usually around 750 ms) between recognition of the cursor hitting the target and the computer recognizing and acting on the command to stop scrolling. This time is far too long for any sort of precision, and precise control of the cursor requires the user to anticipate the arrival of the cursor at the target. Combined with Microsoft Word's behavior of leaving the cursor at the bottom of the screen, this introduces an element of surprise when the desired target finally comes into view, further exacerbating the overshoot problem.

The second variant was to use Microsoft Word's built-in AutoScroll function, which provides a variable-speed auto-scroll controllable by mouse and keyboard (not voice, incidentally). This scroll affects the document, rather than the cursor, and applies a motion blur to the text correlated with the speed of scrolling to make the moving text more visually appealing. However, we found that the motion blur actually makes the text more difficult to read at slower speeds, and at faster speeds, we could not read the text at all. That Microsoft does not allow voice control over the function makes this function useless to those who are unable to utilize the keyboard or mouse.

We decided to replicate Microsoft's AutoScroll function ourselves, omitting the motion blur, with control via speech recognition. We scrolled the document (not the cursor) one line at a time with variable speeds (ranging from 8 lines per second maximum to no minimum speed) controllable by simple voice commands. However, we found scrolling line by line to be too jerky, and as scrolling sped up, the document became more difficult to read. Of course, we again encountered the cursor overshoot problem.

We then switched to auto-scrolling via page-down instead of line down. This scrolled way too fast to read anything, and we rejected it out of hand.

The lesson to the previous three prototypes is that the human eye cannot fixate long enough on moving text to read it. Therefore, our last two prototypes ameliorate this problem by combining auto-scrolling with a pause feature. We auto-scroll the document one page at a time, and then pause for a variable length of time (default pause 2 seconds). The long pause helps the user read the text on the page to discover if their target phrase is on it, and gives them extra time to stop the cursor (before the screen has had a chance to scroll again). However, our use of page-down caused a context problem – the full screen scroll caused the user to lose their place in the document because there was no overlap between the previous screen and the next. In addition, since the page-down function occurs instantaneously, if the user looked away for a second, it might not be easy to tell if the scroll had actually occurred.

We modified that fifth prototype to make the page-down function only scroll 3/4s of the page and to do it over a short period of time (50-100 ms). We implemented at maximum speed of 750 ms/pause to prevent the user from accidentally scrolling too fast.

A final lesson we learned is that once the user has finished scrolling the document, the text cursor must be placed on that screen (which is unlike the behavior of the on-screen scrollbar) and preferably, in the middle of the text (vertically and horizontally), because it provides the shortest distance to any point within the screen, optimizing subsequent on-screen cursor-relative navigation.

4.3.1 Summary of Lessons Learned

1. Cursor overshoot is an inherent consequence of the closed feedback loop with very slow reaction time mainly due to the speed of speech recognizer.

2. When scrolling, do not leave the cursor at the bottom of the screen because the eye will focus on it, and you can not read anything below it.

3. Blurring the text is a counterproductive idea and makes it harder to read at all, no matter what the speed.

4. Microsoft Office (as of version XP) does not allow voice-control over any aspect of the Autoscroll function.

5. Line-by-line auto-scroll appears too jerky to read.

6. Auto-scroll by page-down is too fast.

7. The human eye cannot read text that is constantly moving. The eye must fixate on a word in order to read it[2].

8. Full screen scrolling causes the user to lose their place in the document because they see no continuity of context between the two screens.

4.4 Final Design

Our final design incorporates all of the lessons learned above into a coherent product. The main feature is an auto-scroll and pause, scrolling the text of the document rather than cursor. The scanning speed is controlled by the pause time, initially set at 2 seconds. Each page-down/page-up action is invoked over a period of 100 ms, and scrolls 3/4 of the page, creating a 1/4 page overlap between screens of text. We added line scanning (line up and line down) which operated solely by scrolling and had a variably-controlled speed between 2 lines per second and 20 lines per second. In addition, we added character scanning (left and right) with variably-controlled speed initially set at 8 characters per second. When the user switched from page scrolling to line or character scrolling, the cursor position was place in the middle of the screen horizontally and 1/3 down the page vertically.

We supported nine commands in two categories:

1. **Navigation:** Page Down, Page Up, Go Down, Go Up, Go Left, Go Right

2. **Speed:** Faster, Slower, Stop

Most of the commands are two words/syllables to aid in speech recognition accuracy.

We experimented with a compensation for cursor overshoot. When the user stopped the cursor, we knew its velocity, and could estimate the amount of Δ_{rc}. We used this information to calculate how many units (characters, lines, or screens) we would automatically backtrack the cursor. Unfortunately, in our preliminary experiments, users would "game" the system, trying to estimate the amount of cursor overshoot and anticipating when to stop it, which directly interfered with our naive implementation of this option. It was removed from the product used in the final study.

5 Related Work

Our solutions are inspired by several works: Manaris et. al's SUITEKeys voice-activated keyboard and mouse [12], Igarashi and Hughes's non-verbal voice input [6].

Manaris specifically addresses individuals with permanent motor disabilities (such as those who use a keyboard via mouthstick) and enables them to "press" keys on a keyboard and "move" the mouse by speaking low-level actions. It is not clear whether the voice keyboard has auto-repeat, but the voice can start the mouse cursor moving and then cause it to stop with another utterance. Inspired by this work, we created an ability to start the text cursor in motion and stop it at a later time.

Igarashi enables people to use pitch and volume (instead of speech) to control a button or joystick. We applied this idea to our solution by enabling a user to control the speed of the cursor movement (though we use words to command it rather than non-verbal communications).

Igarashi's earlier work on speed-dependent automatic zooming [5] is also relevant to our work in that he mentions that when a user scrolls too fast, it is hard to read the text, even to just get a sense of where you are. He proposes an automatic zoom-out feature as scrolling gets faster to enable people to gain a better sense of where we are. We believe that our pause concept is superior since stationary text is much easier to read than text moving at any velocity.

Karimullah and Sears [9] studied speed-based cursor control. They recruited non-expert users of speech recognition (none had any visual, heading, speech or cognitive impairments), where we are targeting our work at expert users who are using speech recognition as a primary form of input. Their users, however, did experience the ubiquitous cursor overshoot problem, even though they restricted themselves to a single cursor speed. In addition, our task is more realistic in a work setting, making the user

search through a text document rather than a simple graphical target. Finally, we are working with multi-page documents, which implies that a simple Fitt's law of motion is inapplicable to navigate to the desired target.

6 Implementation

We developed two implementations of our product. Our hardware platform for both was an IBM Thinkpad T20 with a Pentium III running at 700 MHz and 512 MB of RAM, running Windows XP Pro. Both implementations used IBM ViaVoice 9 as the speech recognition software. IBM ViaVoice provides an API to the programmer called SMAPI (Speech Manager API) which enables an application to access voice recognition services, including access to dictation and command and control grammars.

Our first implementation was written in Visual Basic, which we used to script the Microsoft Word 2000 word processor. The speech recognition interface was implemented through IBM's ActiveX controls (provided in ViaVoice 8). The second implementation was written in Java using a modified Swing Stylepad word processor. We interfaced to the speech recognizer via an IBM-provided JSAPI (Java Speech API) plugin.

6.1 Technology Problems

While writing the Visual Basic implementation, we encountered several problems. First, as a computer science graduate student, one author found the Visual Basic language (learned for this project) to be quite a bit harder to understand and use than a more traditional programming language like Java. In addition, the COM OLE Automation documentation for Microsoft Word is poorly organized (alphabetically by function name, even though it is an object-oriented API) and at crucial times, the online documentation was inaccessible. We found that our control of MS Word was superficial and we could not implement sophisticated shading behaviors, which we had intended for a seventh prototype. In addition, all of MS Word's scrolling techniques left the cursor at the bottom of the screen during the scroll, and we did not have enough control over it to move the cursor to a stable screen position.

By contrast, Java's open source implementation and documentation made it possible to work around any difficulties we had in massaging its software to behave the way we intended. However, Java's less than speedy performance prevented us from implementing smooth scrolling behaviors (some of MS Word's scrolling functions were also quite jerky, but it was possible for the most part to avoid them), but due to time constraints, we were forced to accept it.

We eventually abandoned the Microsoft implementation in favor of the Java version, and used it in our voice recognition user study, described in the next section.

7 User Study

In this section, we describe a study of three expert users of voice recognition. We asked each user to perform several editing by voice tasks using their own voice recognition tools and our implementation described above.

7.1 Hypothesis

We predict that the speed of the tasks will improve from the user's own voice recognition tools to our auto-scrolling cursor. In addition, the number of commands spoken should drop, lessening the delay caused by the speech recognizer response time. However, due to the need to control the speed of the cursor, there may be more commands than we predict using our GOMS analysis. We also anticipate the cognitive load of the navigation tasks will go down, as measured the amount of time it takes for the user to scroll the document to the page containing the target.

7.2 Methods

We asked expert users of voice recognition software to perform 8 tasks divided into two similar groups

		# Lines	Total Time (sec)	Multi-page Time (sec)	Reading Time (sec)	# Cmds	# Recog Errors	# User Errors	# Mouse Grid
User 1									
Doc 1	Task 1	87	235	226	74	42	5	3	0
	Task 2	55	75	68	42	5	1	0	0
	Task 3	59	162	150	60	17	3	0	0
	Task 4	92	86	84	48	11	1	0	0
Doc 2	Task 1	88	65	48	22	9	2	1	0
	Task 2	62	129	118	81	9	0	0	0
	Task 3	133	235	208	190	12	1	2	0
	Task 4	154	290	271	173	22	2	4	0
User 2									
Doc 1	Task 1	87	61	58	41	13	1	0	1
	Task 2	55	18	11	10	5	1	0	1
	Task 3	VOID	VOID	VOID	VOID	VOID	VOID	VOID	VOID
	Task 4	92	131	129	85	19	0	1	2
Doc 2	Task 1	88	59	26	N/A	8	0	0	0
	Task 2	62	61	47	N/A	11	0	0	0
	Task 3	133	205	188	N/A	27	0	1	0
	Task 4	154	250	250	N/A	27	0	1	0
User 3									
Doc 1	Task 1	87	69	61	26	16	0	3	0
	Task 2	55	63	51	47	8	0	1	0
	Task 3	59	83	74	58	11	0	1	0
	Task 4	92	100	99	57	23	0	1	0
Doc 2	Task 1	88	33	24	N/A	5	0	1	0
	Task 2	62	153	127	N/A	27	0	4	0
	Task 3	133	98	98	N/A	12	0	4	0
	Task 4	154	145	135	N/A	17	0	1	0

Table 1: This table shows the data collected from the users in our study.
(User 2, Task 3 is voided because he got confused what target he was looking for. Reading times for Tasks 5-8 (using SpeedNav) for Users 2 and 3 are marked N/A because we could not reliably differentiate between the time spent reading and the time spent issuing commands.)

of 4. Each group involved short, medium, long distance, and backwards medium distance navigation through a 10-15 page scene from a Shakespeare play (Romeo and Juliet, Act III, scene 1, and Taming of the Shrew, Act II, Scene 1). One user was familiar with the plays, while the others were not.

Each task was phrased as a search for a specific line in the play. However, learning from our pilot study, we did not give the user the specific words but only a vague description of the line. This approach, we seems to better approximates the kind of navigation task a user is likely to perform on documents that are unknown to him, or those that are not fresh in his mind.

The first group of tasks was performed using the expert users' own voice recognition setup, with their own equipment and software (User #1 used Dragon Naturally Speaking 5.0 on a P2/450 128MB. User #2 used Dragon Naturally Speaking 5.0 on a P3/550 320MB. User #3 used Dragon Dictate 3.01 on a P3/500 128MB). The second group of tasks was performed using our SpeedNav software and laptop that was brought to each session. Users were trained for 10 minutes on IBM ViaVoice using the ViaVoice User Setup Guru. Users then trained for about 5 minutes with SpeedNav on another sample document to gain a feel for our software. We provided a cheat sheet with a list of the nine SpeedNav commands to each user during their tasks.

We video-recorded each session for later analysis.

At the end of the study, we interviewed the participants to gauge their opinions and feelings comparing the two navigation methods.

7.3 Metrics

We measured several quantities (shown in Table 1) to understand the impact of using traditional speech-recognition-based navigation vs. auto-scroll to navigate through a document. We measured the total time per task, time to scroll the document to the page that contained the target (called *multi-page navigation*), the total number of commands spoken, the number of recognition errors, the number of errors caused by misuse of the tool, and number of time the user invoked the mouse grid. In addition, for the users' own speech recognition-based tool, we measured the time spent reading and scanning text (implying that the rest of the time was spent issuing commands and waiting for the response of the speech recognizer).

7.4 Discussion of Results

We compared the two tools using two aggregate measures derived from our data (see Table 2). The first is the number of commands divided by the number of lines traveled. The second is the multi-page navigation time (in seconds) divided by the number of lines traveled.[2]

We ran an unpaired t test that showed no significant difference between the users' own tools and SpeedNav (P = 0.79 for the multi-page numbers, and P = 0.31 for the commands per line numbers), however, the means for SpeedNav were better. We feel that if we re-did our experiment with enough users, we would be able to better discern a difference between the tools.

Another difference between the tools is that users had years of experience with their own tools, but only 10 minutes of practice with SpeedNav. Perhaps, users more practiced with SpeedNav would perform better. From our experience, users dramatically improve their performance with voice recognition software over time.

[2] We use the multi-page navigation time rather than the total time because users universally found that within-screen navigation was much more cumbersome and inaccurate using SpeedNav.

		$\frac{\#\ Commands}{\#\ Lines}$	$\frac{Multi\text{-}page\ Nav\ Time}{\#\ Lines}$
User 1			
Doc 1	Task 1	0.48	2.60
	Task 2	0.09	1.24
	Task 3	0.29	2.54
	Task 4	0.12	0.91
Average		0.25	1.82
Doc 2	Task 1	0.10	0.55
	Task 2	0.15	1.90
	Task 3	0.09	1.56
	Task 4	0.14	1.76
Average		0.12	1.44
User 2			
Doc 1	Task 1	0.15	1.67
	Task 2	0.09	0.20
	Task 3	VOID	VOID
	Task 4	0.21	1.40
Average		0.15	1.09
Doc 2	Task 1	0.09	0.30
	Task 2	0.18	0.76
	Task 3	0.20	1.41
	Task 4	0.18	1.62
Average		0.16	1.02
User 3			
Doc 1	Task 1	0.18	0.70
	Task 2	0.15	0.93
	Task 3	0.19	1.25
	Task 4	0.25	1.08
Average		0.19	0.99
Doc 2	Task 1	0.06	0.27
	Task 2	0.44	2.05
	Task 3	0.09	0.74
	Task 4	0.11	0.88
Average		0.18	0.99

Table 2: This table shows two aggregate measures derived from our data: Number of commands divided by number of lines read, and the multi-page navigation time (in seconds) divided by the number of lines read. (User 2, Task 3 is voided because he got confused what target he was looking for.)

7.5 Post-Study Interview

We conducted an interview with each study participant after completing the tasks. In general, partic-

ipants said that SpeedNav was easier to use than their existing speech recognition system. There were fewer commands required to move to the desired location, and the commands themselves were easy to remember. Participants also appreciated the speed control. One participant also liked the cursor moving by character because he could control the cursor speed to match his reading speed and use the cursor as his pointer.

On the other hand, participants universally had trouble with cursor overshoot, especially when navigating to a location within the current screen. While in general, the cursor overshoot problem is an inherent component of motion-based navigation, SpeedNav exacerbated the problem by not supporting precise positioning within a page. One user wanted to be able to place the cursor at natural landmarks in the document (top of page, top of document, etc). Users observed that relative positioning of the cursor (by motion along the vertical or horizontal axis) is not always the most efficient path to a point on the screen. One user preferred to use mouse grid exclusively to navigate within the screen, and was very proficient at it. Another user felt that if he was more familiar with the document, he would have been able to more effectively use the speed controls, and slow down before reaching the target (minimizing cursor overshoot).

8 Future Work

In our post-study interview, one user expressed a desire for SpeedNav to adjust its speed automatically according to the density of the document text visible on the screen (measured in visible characters). This would be even more interesting if SpeedNav controlled the initial speed of motion when beginning a navigation task as well. This would provide an alternate means of adapting to the user's inherent reading speed than provided by Igarashi's zoomable user interface [5]. Both controls attempt to preserve a constant density of text per unit time.

Another method to speed up the user's navigation performance is to improve the user's own technique in scanning text. A course in speed reading might nicely complement our SpeedNav work.

In addition, we think that if we add a shaded region of lines to the document it could make these lines easier to read. The idea is to shade three lines of text with a pastel background, and when the auto-scroll is active, we move this shaded region down the page, one line at a time, until it reaches 3/4 of the way down the screen (it will make this journey within the pause time for reading this page). When the shaded region hits this point, the entire screen will scroll 1/2 a page (leaving the shaded region 1/4 from the top of the screen). This shaded region will help draw the eye (from the cursor) and focus the reader to scan the text from top to bottom as the shaded region moves.

We feel we can improve the sophistication of the cursor overshoot correction algorithm in a novel way. Since $\Delta_{rc} > 0$ requires that the cursor will go further than where the user intended, we propose to place two shaded regions (of different colors) on the screen. The first shaded region goes at the top of the page (when the user is scrolling down, and at the bottom when scrolling up. For the rest of this example, we will assume a downward scroll), and the second, $\Delta_{rc} \times speed_{scroll}$ lines below it, centered in the middle of the screen. When the user scrolls, they will read the text in the center shaded portion, but in reality the system assumes the "cursor" is in the upper shaded portion. When the user says "stop", the "cursor" in the upper shaded portion scrolls down to the center shaded region and stops, eliminating the users' perception of the overshoot. This kind of trickery was studied for cursor motion within a screen towards a graphical target in [9]. Users experienced higher error rates with such an automatically correcting cursor, but their experiment used no speed control, which we feel might enable users to slow down to a comfortable reading speed.

In a completely different direction, we brainstormed two new techniques for document navigation that we did not have time to implement. The first is context-aware mouse grid, and the second is a phonetic Google-like search tool.

8.1 Context-Aware Mouse Grid

The beauty of mouse grid is its simplicity and speed in quickly moving the cursor to a precise location on the screen. However, it does not "know" what is on the screen. In addition, people do not and cannot remember unique terms for all semantically important areas visible on the screen (even worse, icons usually have no text alternative). If mouse grid could be context-sensitive and be able to tag semantically important areas of the screen, such as the menu bar, toolbar, scrollbar, document with textual names or numbers, then, when the user invoked mouse grid, visible tags would appear on the screen naming the area of the screen which can be zoomed in upon. The zoomed-in mouse grid would then tag more important areas within that initial area (such as each individual menu, or each section of the toolbar, or the large areas of the scrollbar (thumb, up arrow, down arrow, page up section, page down section) etc. If the user zoomed in on text, the text would be annotated with numbered tags to visualize the paragraph, then sentences/lines, then words (and characters if necessary) to enable the user to precisely drill down and move the cursor where they want it to go.

8.2 Phonetic Google-search

To better adjust for speech recognition errors in the find tool, we propose that for voice-enabled find, the tool use phonetic searching rather than exact character matches. Also, the order of the words should not matter, making the search similar to Google, rather than traditional word processor search. In addition, we feel that this find tool should be made non-modal, to eliminate extraneous dialog box interactions (bring up the dialog box, click OK to dismiss, click Find Next to continue the search) which slow down voice control. In addition, the search tool should bring up all results of the search and display them in summarized form (with a sentence of context above and below) on one side of the screen. Each match would be numbered and the user just say the number of the desired match to go to the location.

9 Conclusion

Document navigation, the less glamorous aspect of speech recognition, deserves more attention from the research (and commercial) community. An improvement in this functionality will enable those with motor impairments to enjoy the same ease of editing that non-impaired people take for granted.

This work contributes to our understanding of the performance of the current state-of-the-art when used by people with motor impairments. We have shown through a GOMS analysis that the only ways to improve this performance are to reduce the number of commands and the cognitive load on the user. Reducing the latency in speech recognition will help, but the problems will not go away until speech recognition response is as fast as a keyboard or mouse. Our SpeedNav tool showed the potential to reduce the number of commands and the cognitive load through an auto-scrolling mechanism. Even though our results were inconclusive (comparing SpeedNav to commercially-available solutions), further development along these lines (as well as a larger user study) should show a more significant result.

10 Acknowledgments

We would like to thank all the participants in our user study and the members of the Assistive Technologies class in which this research was conducted. In addition, Prof. Jennifer Mankoff gave us valuable advice and feedback on this work.

References

[1] Johnny Accot and Shumin Zhai. Beyond Fitts' law: models for trajectory-based HCI tasks. In *Proceedings of ACM CHI'97 Conference on Human Factors in Computing Systems*, pages 295–302, 1997.

[2] S. K. Card, T. P. Moran, and A. Newell. *The Psychology of Human-Computer Interaction.* Lawrence Erlbaum, Hillsdale, 1983.

[3] P. M. Fitts. The information capacity of the human motor system in controlling the amplitude of movement. *Journal of Experimental Psychology*, 47(22):381–391, 1954.

[4] IBM. ViaVoice Speech Recognizer. http://www-3.ibm.com/software/speech/.

[5] Takeo Igarashi and Ken Hinckley. Speed-dependent automatic zooming for browsing large documents. In *Proceedings of the ACM Symposium on User Interface Software and Technology*, Speedy Input, pages 139–148, 2000.

[6] Takeo Igarashi and John F. Hughes. Voice as sound: Using non-verbal voice input for interactive control. In *Proceedings of the 14th Annual Symposium on User Interface Software and Technology (UIST-01)*, pages 155–156, New York, November 11–14 2001. ACM Press.

[7] Bonnie E. John and David E. Kieras. Using goms for user interface design and evaluation: which technique? *ACM Transactions on Computer-Human Interaction (TOCHI)*, 3(4):287–319, 1996.

[8] Clare-Marie Karat, Christine Halverson, John Karat, and Daniel Horn. Patterns of entry and correction in large vocabulary continuous speech recognition systems. In *Proceedings of ACM CHI 99 Conference on Human Factors in Computing Systems*, volume 1 of *Speech and Multimodal Interfaces*, pages 568–575, 1999.

[9] Azfar S. Karimullah and Andrew Sears. Speech-based cursor control. In *Fifth Annual ACM Conference on Assistive Technologies*. ACM, 2002.

[10] Lewis R. Karl, Michael Pettey, and Ben Shneiderman. Speech versus mouse commands for word processing: An empirical evaluation. *International Journal of Man-Machine Studies*, 39(4):667–687, 1993.

[11] I. Scott MacKenzie and William Buxton. Extending fitt's law to two-dimensional tasks. In *Proceedings of ACM CHI'92 Conference on Human Factors in Computing Systems*, pages 219–226, 1992.

[12] B. Manaris, R. McCauley, and V. MacGyvers. An intelligent interface for keyboard and mouse control: Providing full access to pc functionality via speech. In *FLAIR*, 2001.

[13] Microsoft Corp. Microsoft Office XP. http://www.microsoft.com/Office/.

[14] J. Sachs. Recognition memory for syntactic and semantic aspects of connected discourse. *Perception and Psychophysics 2*, 1967.

[15] Scansoft Inc. Dragon Naturally Speaking. http://www.scansoft.com/naturallyspeaking.