AbleControl: Computer Navigation for Fine Motor Control Disabilities

Mona Abdelrahman

Introduction and Overview

One of the biggest problems with creating new technologies is having them be accessible to people of all abilities. For people who need accommodations, the lack of accessibility tech results in people having their interactions with technology limited, the technology being hard to use/understand, or simply being completely alienated from new tools.

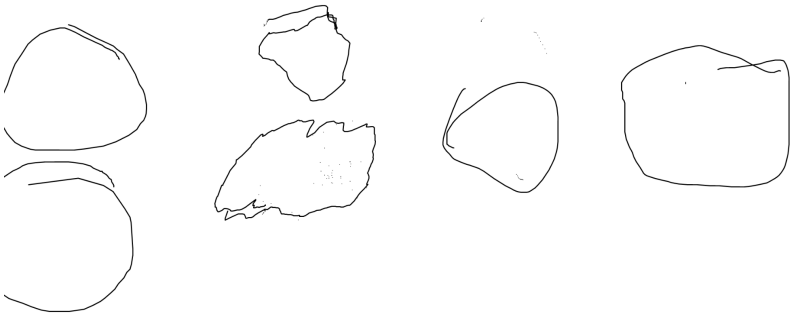
In my project, I am focusing on creating a motion-based system for people who have fine motor control issues (that manifest in hand tremors). For people with fine motor control issues, it is often hard for them to use a keyboard and mouse because of the smaller, precise movements that are required for use. I wanted to create a system that allows for full computer interaction and isn't regulated to a specific program or scenario. I also wanted a system that is somewhat familiar in terms of the user interface and the interactions. This motivated me to use the LeapMotion sensor to allow for a large gesture-based input. Since these individuals have issues with small muscles, using larger muscle groups (such as those controlling whole hand movement) allows for them to still have the familiarity of hand interaction while still accommodating for their needs. Another reason why I decided to use gesture as the primary input mode is because using a primarily voice-based system would be susceptible to background noises or if other people are around. Similarly, eye tracking can be hard to use when there is uneven lighting or when the user is wearing glasses. Although my system uses a combination of gesture and voice, the voice system is primarily for inputting text and some verbal action shortcuts.

Beyond just accessibility, this can also be helpful for use for anyone who also might not be able to use their hands in a full capacity due to injury or circumstance (ex. sticky hands).

System Description

The current system takes in two inputs: hand tracking/motions and voice input/controls. The hand controls serve as a way to replace the mouse, and the voice input replaces the keyboard. Currently, the system moves the mouse cursor on the screen of the user's device when the user's hand moves. For example, if the user moves their hand up, the cursor will follow and also move up. The user can also perform clicks, double clicks, and click-and-drag motions by placing their hand into a grabbing position. If the user holds this position for 2 seconds, they perform a single click. If they hold for 4 seconds, they double click. If they hold for 6 seconds, they are in click-and-drag mode until they release the grabbing motion. The user can also perform a right click by performing the grabbing motion with their palm facing up for about half a second.

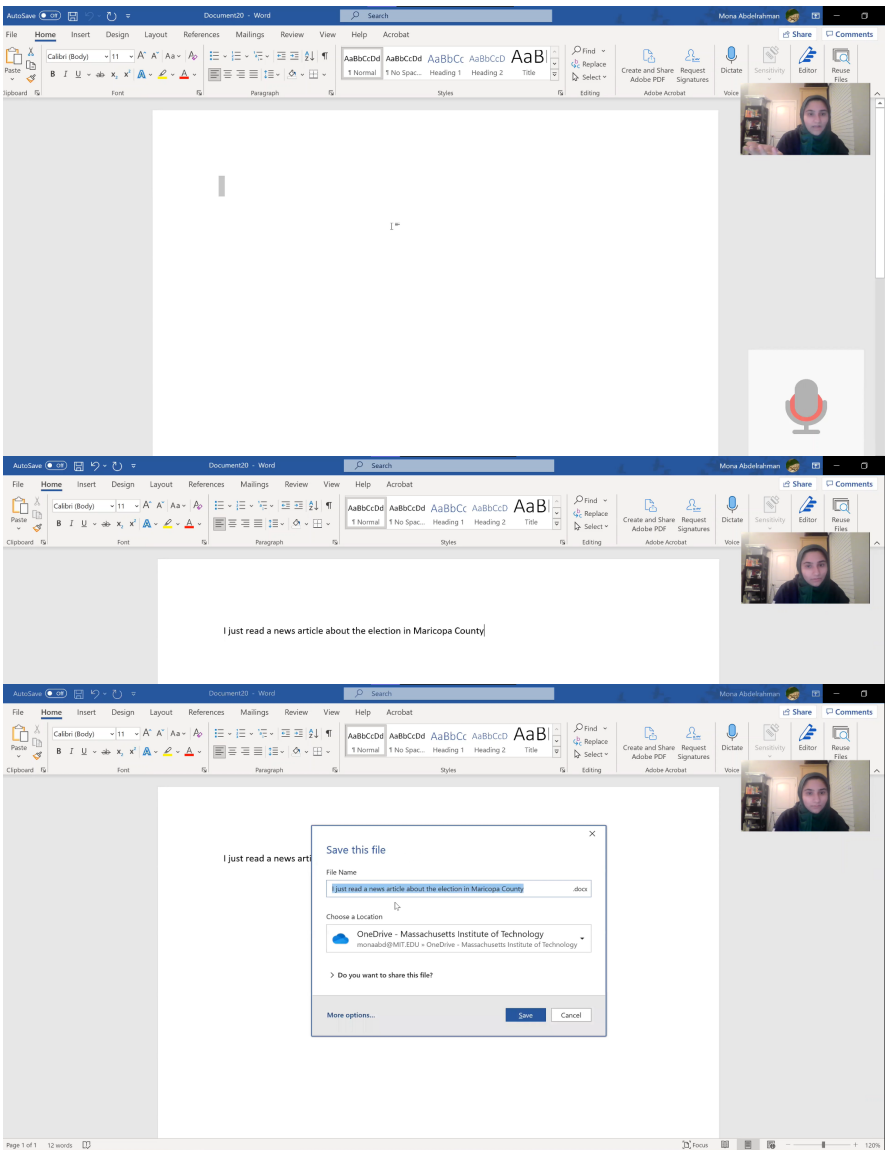
To input a phrase into an input field, the user simply has to click on it, turn their hand so their palm is facing up, wait a few seconds, then speak their phrase. The phrase will then appear on the screen. For the user to input a voice command, they will perform the same action and use one of the pre-defined voice commands instead. These commands include actions for saving a file, opening a file, closing a window, undoing an action, redoing an action, and the usage of some keyboard keys (such as enter or backspace).



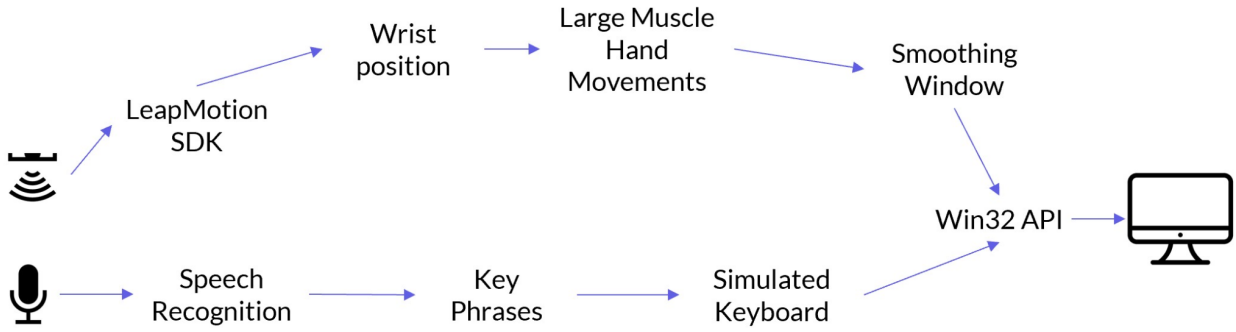
With mouse | System, no smoothing | System, smoothing | System, smoothing, with shaking hands

Here on the left, we can see some sample inputs from the click and drag motion in paint. In the far left, I have an example of drawing circles with a standard laptop trackpad. Next to it are some circles that I drew with the system, but without any smoothing. As we can see, it is very jittery. In the middle right, I have an example of a circle I drew with the system and smoothing activated. As we can see, the smoothing does a great deal for reducing the jitteriness of the LeapMotion tracking inputs. On the far right, there is another circle drawn with the system, but with shaking hands to simulate a hand tremor. In this case, we can see how effective the smoothing is at reducing the impacts of hand shakiness to prevent unwanted motions and behaviors with the mouse.

The example shown on the right is demonstrating the usage of some of the voice control features. In the very top frame, I have a Word document open and I am placing my hand in the upright position to enter in some text. The microphone indicates that it is active by displaying an icon in the bottom right corner so the users can know that the system is ready for them to input text. In the middle frame, we see that the system recognized the audio and placed the text into the document. In the last frame, I opened the microphone again, and said the key phrase “save file”. As shown, the system opened the ‘save file as’ dialog box.



How it works



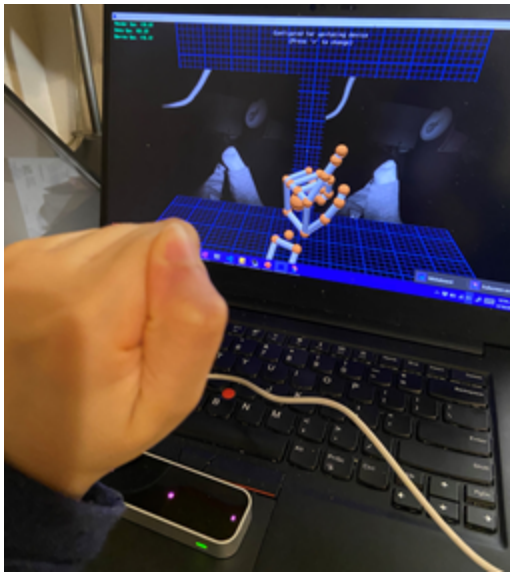
In the diagram above, we can see a general system architecture and overview. The LeapMotion sensor interfaces with Python by using the LeapMotion SDK. From the LeapMotion SDK, I decided to use the wrist position function instead of the hand position as I had originally intended. A core design principle for the gestures and movements was to have large muscle group movements and gestures to fulfill the project goal of being an accessibility system. Hand movements and gestures are then passed through a smoothing window of about 40 frames that average out the location of the hand. The change in position from the last location is then calculated and passed to the Win32 API to interface with Windows. This API allows for Python to interface and control aspects of the system state in Windows.

For the speech input, it requires that the user have a microphone input set up on their computer. The system then recognizes the voice input gesture (flipping the hand over) via the LeapMotion sensor by getting the palm coordinates. Once the user activates the speech recognition module, they can either say key words for shortcuts or say a phrase to enter text in a field. All key phrases are recognized, and if no key phrases are present, the system sends text to the computer. The key phrase actions are done by simulating keyboard shortcut entries by sending the corresponding keyboard shortcut to Windows (as a keypress combination) when a key phrase is detected. For text input, the system interfaces with the windows keyboard to "write" text to the field that is selected.

As previously mentioned, the core design principle when choosing how to make gestures was to have primarily large muscle hand movements. Afterall, the main goal of the system was to be an alternative input mode for users with fine-motor control issues, so being able to avoid these as much as possible is a cornerstone of the project. Because of this, I opted to use the wrist position instead of the hand position as I had originally intended. The main reason behind this is because the wrist position is much more stable than the hand position, so this allows for the system to be more accurate when it comes to hand tracking and it reduces the potential for unintended movements by the user. The two main input gestures are grabbing to click and flipping the hand over to input text. Flipping the hand over requires larger arm muscles, so it should not be impacted by any hand shakiness or tremors. The flipping motion was actually simple to implement, since it only required getting the palm normal vectors and requiring that the

y and z coordinates fulfill certain conditions (mainly that the y normal is above 0.7) to be detected to be in the up position.

When deciding on the clicking motion, I opted to go for one of the predefined motions in the LeapMotion SDK - grabbing and pinching. This was done because I wanted a motion that simulated the physical presence felt when someone would click, as this is a very subtle form of feedback. However, I also wanted the motion to be able to be done with only one hand, and it's important to have a motion that cannot be mistakenly recognized as another (such as a clap, where the user has to move both hands, which can cause other unintended consequences). When deciding between grabbing and pinching, it ultimately came down to which one is easier for people with hand tremors. Although my testing revealed that pinching was more accurately recognized by the sensor, it is extremely difficult to maintain a pinching motion for the time period required to do a click, much less any of the other clicks that require holding the position for longer periods of time. The pinch motion requires the thumb to touch the fingers, which cannot be held consistently with hand tremors. Conversely, the grabbing motion is slightly less accurate, but it relies on having a general grabbing position, and the hand does not need to be in a tight fist. Testing with the grabbing motion shows that as long as the hand is in a general curled position, any hand tremors/shaking do not significantly impact detection. This makes the grabbing motion the overall best motion to combine the UI feedback needs and the accessibility needs of the users. However, the grabbing motion isn't perfect, as we can see here on the left. In this picture I have my hand in a tight fist and the leapMotion sensor is processing one of my fingers as sticking out. Although it can be resolved by removing the hand from the frame and bringing it back, it can be annoying at times.



As explained above, the clicking functionality was one of the trickiest parts of the project, since it required doing a lot of testing to figure out timing for each position and gesture modification. Testing usually consisted of navigating within a specific program/the desktop and seeing if I can click around without too much trouble. This was the part of the project that I iterated over the most. The click and drag motion is also inconsistent, as it works well in the context of stroke drawing in paint, but doing the exact same function in the OneNote desktop app did not work. This may be because of some privileges that are not allowed access by the Win32 api for security reasons. Although I hoped to be able to resolve this issue, it was more obscure than I initially thought. The challenges with the click and drag functionality taught me how oftentimes things can be grossly simplified when discussing features and design. In the context of this project specifically where I am trying to replace another device, it often is not enough to

have analogous substitutions for functionality and may require thinking more about the actions in terms of their intended consequences instead of the action itself.

However, the voice recognition was one of the more simple aspects of the project to implement. The hardest part with the voice input was actually having the UI icon show up properly when the microphone was active and deciding on a voice command schema that was consistent to not confuse the users. To test the voice commands, I used them in the context of them being used in the middle of other tasks, such as clicking somewhere and entering text or using the commands to compliment the mouse functions. Since I used a third party voice recognition API, I mainly tested the voice to simulated keyboard pipeline to ensure that it was performing the expected behavior. The smoothing and cursor movement was also fairly straight forward, it mainly required testing to tune internal parameters. These tests involved simulating hand shakiness and seeing how the system was reacting, and also testing the cursor responsiveness when gesturing.

Overall, the system works well for cursor and voice movements, and single clicks work well enough. Double clicks and right clicks also have decent performance, although not as consistent as the single clicks. The click and drag motion is the most inconsistent, but it allows the user to scroll, which is the biggest use for it. For general system testing, I would perform tasks such as clicking on small buttons on the screen, opening programs, entering text, and performing actions within the programs (such as opening and saving files).

Project Modifications + Roadblocks

The project features and goals have gone under significant modifications since the Design Studio. Originally, I had planned for there to be motion correction and motion prediction. However, as I implemented the smoothing, I realized that the motion correction became redundant since the primary motion correction would be hand shakiness, which the smoothing already takes care of. With the motion prediction, it would not have been a good UI, since if the users wanted to have suggested motions, they would still need to navigate such a UI by using gestures, making it redundant at best.

I had also originally planned to have an onscreen keyboard so users can virtually type; however, this was quickly scrapped after it became obvious that it was too tedious to enter in each key individually and the clicking motions did not interface well with the users and the on screen keyboard, since it resulted in a lot of accidental clicks on the keyboard, making it effectively unusable. I got around this by using the voice input previously mentioned. Using speech recognition was much more effective for user input and action shortcuts/commands.

User Studies

Although I did not have the opportunity to test the user studies with people who have fine-motor control disabilities, I was able to do some testing with members of my immediate household (my sister and my parents). I noticed that while it took a while to understand the movement plane of the hand, once it was understood (usually by showing an example) it was

fairly simple. The user studies also helped guide how to display the instructions page and some smaller points of user feedback-such as the range of the cursor and adjusting parameters. The user studies were also very helpful in guiding what sort of features I should add. It made me realize the gaps in the system for usability and what things I had taken for granted. As a result of the user studies, I added in a start screen so users can decide on a primary input hand, allowed for two hands to perform input at the same time, and removed the on-screen keyboard. The user studies also helped me visualize the progress from really unusable to fairly usable at the end. This was valuable as oftentimes I was not always able to tell how I was progressing, and seeing someone else use the system allowed me to shift focus to the needed areas.

Performance

Overall, the performance of the system is decent and users are able to navigate and do basic tasks. Although there is some inconsistency with the clicking actions, the smoothing does a lot to help with shakiness and performs quite well. The smoothing works by taking the average of the last 40 LeapMotion frames. Since hand tremors involve the movement of the hand in an oscillatory movement, smoothing by averaging effectively takes the position that the oscillations are occurring around. The biggest takeaway from the testing and overall performance was that the things I expected to be the most simple where actually the most difficult to iterate over (mainly the clicking), and the items I thought I would spend the most time on ended up to be much more simple in nature (the motion smoothing). In general, I think the clicking isn't as robust as I would hope because it is hard to anticipate how long of a cool down period different people may need between clicks, although I have found about 2 seconds is a window that works for most people (without the window being too long or too short).

I think something that is just out of reach is consistent click-and-drag functionality. Although it works in some cases, it doesn't always work, which should be a simple fix in theory, but it is moreso finding the right area where the click-and-drag is breaking. Once this is resolved, I believe that the users will have almost full functionality of the computer and be able to have access to everything that is possible with a keyboard and mouse.

The next steps to improve the system would center around allowing for more than just premade functions to be made. Currently, the voice commands work by sending keyboard shortcuts to the OS, so I think it would be really helpful to be able to verbally say any keyboard shortcut and have it automatically convert to the correct key presses, without it needing to be predefined. I would also like to find a way to be able to add punctuation in sentences seamlessly. An example of this would be saying "Hello comma my name is Mona period" so the text 'Hello, my name is Mona.' appears on the screen. Although it seems simple enough, I think there may be several use cases where this can go wrong and be frustrating for users so the UI may need to be reworked to account for different cases, such as wanting to write the word 'period' instead of '.'. These additions would make the system more flexible for different users and make it simpler and more enjoyable to use.

Conclusion

Overall, the system seems to be a good start for a general gesture-based replacement for the keyboard and mouse. However, the system still has some interesting quirks--some as a product of the LeapMotion sensor, some from the surface-level limitations of the Win32 API for Python. Within the project I learned a lot about the process of ideating, designing, and implementing a system from scratch. It was also valuable to have things fail and not work as expected. One of my biggest lessons was that your users may ideally act a certain way, but reality can be much different. Although it is not perfect, there is a functional skeleton that allows for further improvement and features.