MusicBuddy

Becky Bell, Elaine Lin, Phoebe Tse
6.835 Final Project

Task and Motivation

Many casual guitar and ukulele players will often look up the chord charts to their favorite songs online, usually on a desktop computer. However, the entirety of these chord charts do not completely fit within the viewport of the browser, so these musicians constantly find themselves interrupting their playing in order to scroll down and see the rest of the song. This can become quite cumbersome, especially if you use online chord charts often. Our goal was to develop a system that could track a musician's progress in the song and scroll for the user, instead of the user having to scroll for themselves. Like someone who turns the page for a pianist during a recital, MusicBuddy is a multimodal interface that tracks a ukulele player's progress in an online chord chart and scrolls down for the user when needed.

MusicBuddy

MusicBuddy takes in three modes of input. The first is speech recognition, which the system uses to track the current lyric line the user is singing. The second mode of input is motion detection, which the system uses to track the current chord line the user is playing. More specifically, the system tracks when the user's left hand on the fretboard moves, indicating that the user is switching to the next chord. The third and final mode of input is gaze tracking. The system uses this input to roughly determine where on the page the user is focusing.

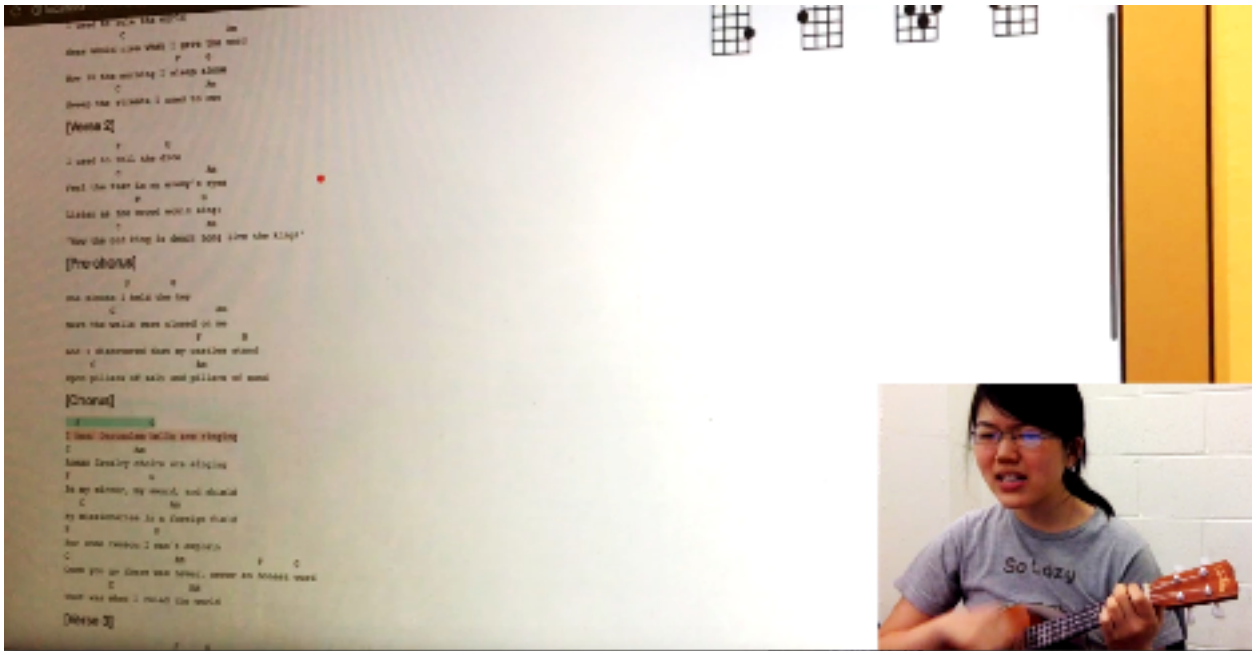

Figure 1. An example of the MusicBuddy system. The user sings and plays along to a chord chart while MusicBuddy tracks the current lyric and chord line (indicated by the highlighted lines on the screen) as well as the user's gaze (indicated by the red dot).

The user presses the ‘Start’ button and begins singing and playing. Based on all three of the inputs mentioned above, the system will estimate the current y-position on the screen corresponding to the user’s place in the song. Based on this y-position, the system will either scroll faster or slower to match the user’s pace.

System Overview

As mentioned above, the MusicBuddy system uses a combination of motion detection, speech recognition, and gaze detection to estimate when the user needs faster or slower scrolling.

A list of the specific libraries are included in the Tools section. Figure 2 below illustrates the system architecture and how we went about fusing these three modalities.

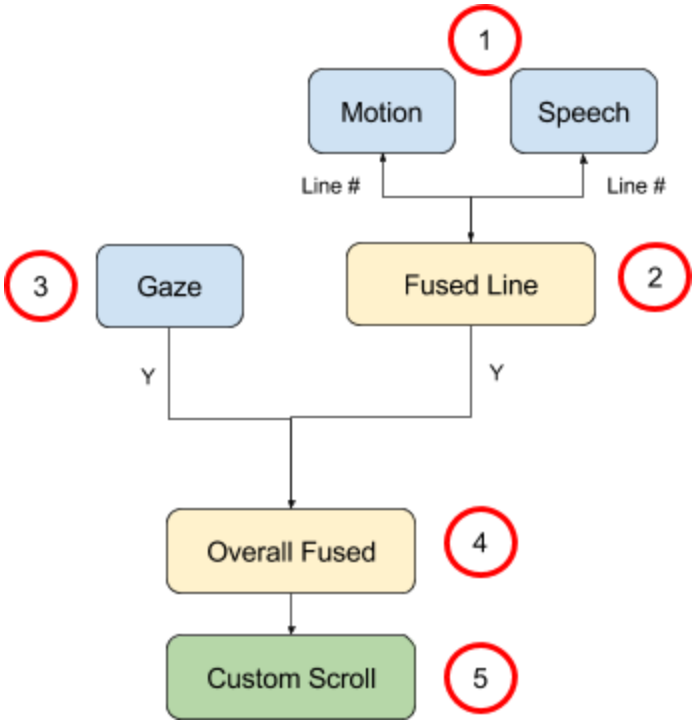

Figure 2. System diagram for MusicBuddy

A song has multiple lines where each line is a chord-lyric pair. Figure 3 shows an example line for “Viva la Vida.” MusicBuddy uses the Mustache templating language to render each song and index each line with a line number.

Figure 3. Example of a line in the song "Viva la Vida." A line is a chord-lyric pair.

Let *Line #* represent the current line that the system thinks the user is currently singing. Let *Y* refer to a decimal value measured from the top of the window, representing the system's understanding of the user's current y position. *Fused_Line* is the function that fuses the line number outputs from motion and speech. *Overall_Fuse* is the system's fused understanding of the user's current y position, considering all three modalities. *Custom_Scroll* is a function that determines the rate based on the output from *Overall_Fuse*.

Motion and Speech Detection
Motion detects any significant motion using the webcam. For our purposes, we modified the motion detection library diff_cam_engine to only detect motion in the bottom left input (the user's left hand) in order to yield better results in detecting chord changes along the fretboard.

When either the motion or speech detection detect that the user has advanced to the next chord or lyric line respectively, the modalities call a function that fuses these outputs into one fused line number. Each modality outputs a line number, and are fused as a weighted sum, rounded to the nearest integer.

Fused_Line
This function takes in the current line that each modality thinks the user is on, and then outputs a weighted summation of the two. This sum represents the fused line number- the system's guess of which line the user is currently on given the motion and speech modalities.

$$line_{fused} = c \times line_{motion} + (1 - c) \times line_{speech}$$

where $line_{fused}$ is the system's best estimate of the user's actual current line. $line_{motion}$ and $line_{speech}$ are these modalities' individual estimates of the user's current line. c is some constant between 0 and 1 that indicates the amount of weight that the motion detection has on the final fused line calculation.

This function returns the fused line's corresponding y position with respect to the top of the browser.

Gaze Detection

Every few seconds, gaze outputs the y coordinate the system thinks the user is looking at every few milliseconds. We smoothed this output by taking the average of the y coordinates over a predetermined number of seconds. The system takes this estimate and calls a fuse function to determine if the estimated fused y position of the user is outside of the ideal region (See Figure 4).

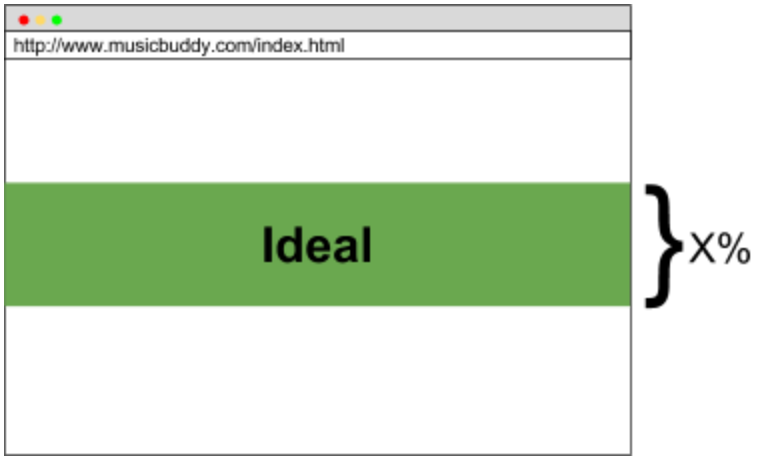

Figure 4. Scroll rate on the MusicBuddy interface depends on where the user's current position is on the web page. The middle region of the web page is ideal.

Overall_Fuse

This function takes in the Gaze modality's estimate of the user's y position and the Y position output from Fused_Line function. Then, it calculates a weighted summation of these two y positions to give a final fused Y. This represents the system's best estimation of where the user is currently positioned on the screen, using all three modalities.

$$Y_{Fused} = \frac{w \times Y_{FusedLine} + (1-w) \times Y_{Gaze}}{2}$$

Y_{Fused} is the system's best estimate of location on page based on all three modalities. $Y_{FusedLine}$ is the Y position of the Fused Current Line's HTML div. Y_{Gaze} is an average of Y coordinates of the user's gaze over the past few seconds, determined by the Gaze modality. w is some weight between 0 and 1.

Custom_Scroll

Our system scrolls at a constant rate, regardless of where the system believes the user is. This function takes as input Y_{Fused}, the system's estimate of where the user is on the page, and

determines how much more to scroll. More precisely, this function determines the change in the number of pixels of the default scroll amount.

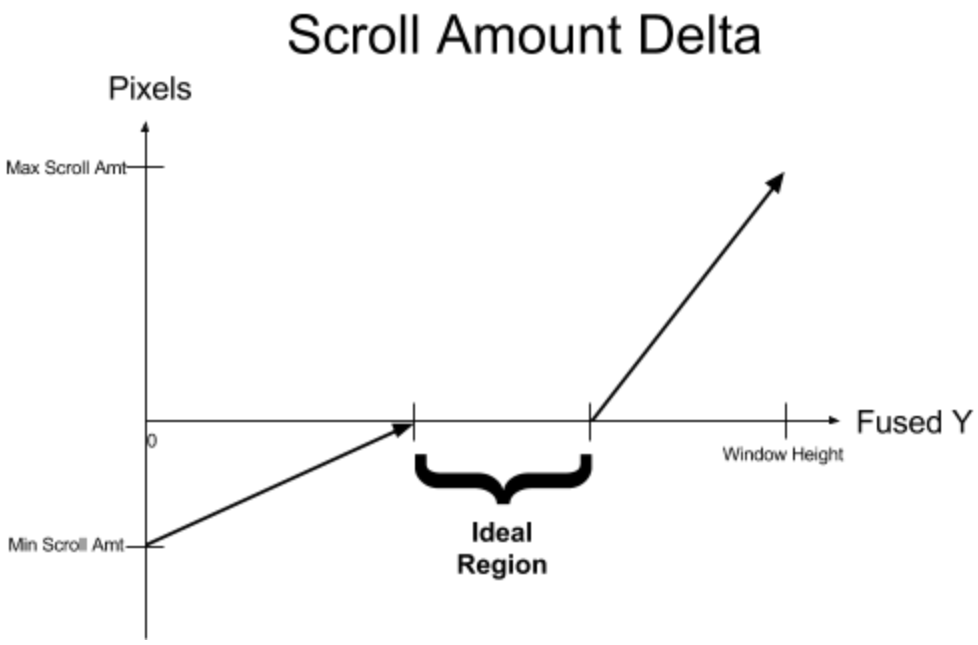

Figure 5. Scrolling rate (pixels) depends on estimated position of the user's location on the screen (fused Y)

In Figure 5, the X-Axis is the different values that the system may output as the Fused Y. In theory, this could be anything from 0 (the top of the window) to the height of the window itself. The Y-Axis represents the number of pixels more or less than the default amount that the system should scroll. For example, when the Fused Y is in some predetermined ideal region, the system would continue to scroll at its default scroll rate (i.e. delta = 0).

The arrows represent how the scroll amount changes depending on the Fused Y value. For instance, if the system thinks the user is all the way near the bottom of the window, it would scroll larger amounts (a faster rate) in the effort of bringing the user's current view closer to the ideal region. Generally, the greater Fused Y is, the lower on the page the user is at, and thus the faster the system would need to scroll. Likewise, the lower the Fused Y is, the higher on the page the user is at, and thus the slower the system would need to scroll.

Note that it is possible to change the parameters here such that the system scrolls up if the user's current view is too high. In our user studies, however, we decided to configure the parameters such that scrolling up was not possible. It was better for usability if the system never scrolled up,

because if the system lagged behind the user, the system would perpetually scroll upward, as we saw in our pilot tests.

System Performance

The goal of the MusicBuddy interface is to scroll automatically on a chord website so that the user can follow along with the song without taking his or her hands off the ukulele to scroll. During the user studies, we measured performance by counting the number of interruptions. An **interruption** is when the user take her hands of the ukulele to adjust the view of the web page.

The MusicBuddy interface performed poorly in the pilot tests. The interface severely lagged behind the user. Concerning scrolling, it barely scrolled down and sometimes scrolled up to compensate. The poor performance happened because the room used for user testing differed from the room used for development. The Webgazer performance depends heavily on the lighting conditions of the room. The room used for development was dim, and the gaze modality did not work well, so we decreased the weight of gaze in fusion. However, the room used for user testing was brightly lit, and the interface could track gaze, so we increased the weight of gaze in fusion. Changing the weight of gaze required adjusting other parameters, so many parameters changed post-pilot testing.

The MusicBuddy interface performed well in the remaining four user tests. The good performance happened because we changed the parameters specifically for the room used for user testing.

The changes in parameters are shown in Table 1 below.

	Pilot Testing Parameters	Post-Pilot Testing Parameters
Motion Detection Engine		
PIXEL_DIFF	80	40
SCORE_THRESHOLD	15	30
Line Fusion Weights		
MOTION_W	0.7	0.5
SPEECH_W	0.3	0.5

Y Fusion Weights		
LINE_W	0.95	0.6
GAZE_W	0.05	0.4
Scroll Fusion		
SCROLL_AMOUNT_DEFAULT	1 pixels	7 pixels
MAX_SCROLL_DELTA	300 pixels	700 pixels
MIN_SCROLL_DELTA	-1 pixels	0

Table 1. Parameters used for determining scrolling before and after the user study pilot tests.

PIXEL_DIFF is the measure of sensitivity the motion detection engine can have. SCORE_THRESHOLD indicates how high of a 'difference score' that detected motion needs to have before our system deems it a significant amount of motion.

SCROLL_AMOUNT_DEFAULT is the number of pixels the system scrolls per second, regardless of where the user is currently in the song. MAX_SCROLL_DELTA is a parameter used in our function that outputs the number of pixels to scroll based on the system's detected Y position.

Changing the MIN_SCROLL_DELTA parameter to zero makes sure that the system does not scroll upward. Previously, this was the case when the user's position was higher up on the page than the system thought they were at.

In addition to these parameter changes, we addressed user feedback and added more white space at the bottom of the tabs so that it is more obvious to users when they have reached the end of the song. Some users also asked to see the calibration webcam output while they are playing. Thus, we decided to give the user the option to toggle the calibration webcam output and placed it to the right of the lyrics.

See the User Study Recap section for more details on our user studies.

Implementation Challenges

Throughout the implementation process, our opinions of each modality changed. At first, we thought gaze would be pretty reliable. Later on, we realized it was not very reliable. However,

during our user tests, we realized it was actually pretty accurate for our purposes when we were under a good amount of lighting.

At first, we knew that speech would be a bit unstable from our experience with the 6.835 Battleship mini-project. However, at one point during the implementation process, speech detection performed decently well when singing and playing the ukulele with good enunciation. By the time of the user tests though, we realized again that speech was not very reliable.

We knew within the first 2 weeks of implementation that the Leap Motion device was not sufficient for our purposes. We had intended to use the Leap to track motion between chord changes on the ukulele fretboard. However, once we started testing, it was clear that the Leap was having a lot of trouble detecting hands when they are holding an object and facing upwards. As a result, we pivoted and found a general motion detection javascript API that uses the webcam. This worked as a good replacement for the Leap.

Speech was more difficult than expected. The difference between singing and speaking was not an issue, but instead, the issue was both speech detection and word recognition. Detection was overall difficult without a microphone, and even then, the WebSpeech API would sometimes stop working and only start to work on browser restart. In the end, we decided to implement the speech modality to have a very rudimentary way of checking if a lyric line was stated: if at least 3 words in a lyric line were detected, the system would move on to the next line.

Relatedly, song lyrics often repeat, and so, speech detection was a little more difficult than expected because with repeated lyrics and laggy detection, the system would sometimes get way ahead of the user.

Motion was more difficult than expected as well. The webcam output we had was fairly accurate at detecting all ranges of motion- small and large- depending on parameters. Extracting out what was considered 1 chord change on the fretboard, however, was a bit more difficult. Empirically, we found that 1 chord change is generally equivalent to two video frames that consecutively display significant motion (the library captures webcam data every 100 ms).

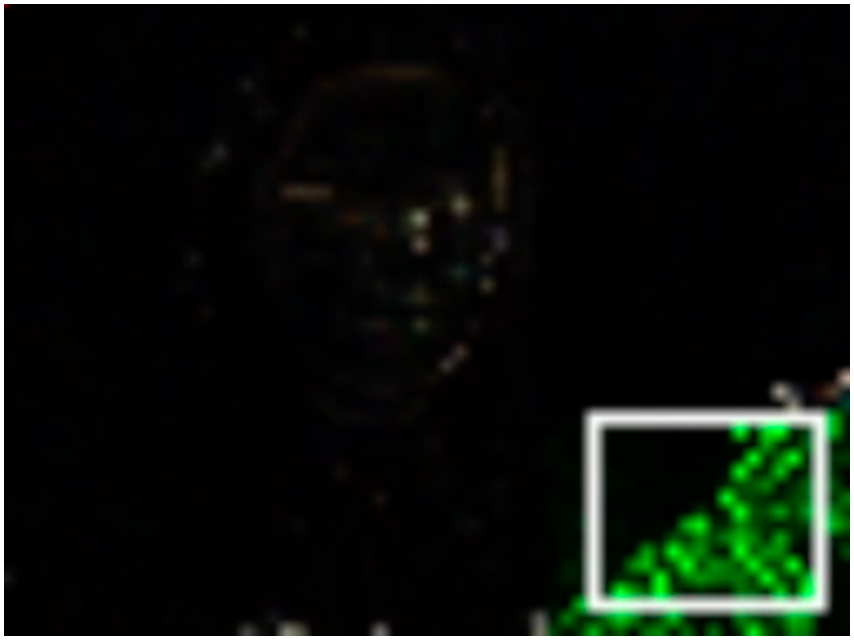

Figure 6. Motion Detection Webcam Output. The white box indicates that significant motion was detected. The green pixels indicate that some motion was detected.

Gaze detection worked fairly easily. The WebGazer.js library was very easy to set up, and the API was clear enough for our simple purposes. As long as the user calibrated on the page well by clicking with the mouse, this library was pretty accurate- even for people with glasses. The only issue was the library's dependency on good lighting. For example, during our user tests, we had sufficient light, thereby leading to fairly accurate gaze detection results.

Roadblocks and Pivots

Aside from the pivot we made in the beginning from Leap Motion to Webcam (discussed in the previous section), we had another roadblock that led to our second "pivot".

Initially, we had intended for motion and speech detection to be the baseline modalities that would have very heavy weight in the final fusion, while gaze would have relatively less weight. The idea was that, if debug mode was on, the user's current line in the song would be highlighted and would move whenever the user moved to the next one. However, we realized that motion detection was either always too sensitive and got ahead of the user, or not sensitive enough. At this point, we also realized that gaze detection was more reliable, especially under good lighting. Therefore, our pivot was to embrace the fact that motion detection was very fast and allowed gaze and speech to balance out motion.

As a result, when debug mode is on, the highlighted lines generally get a few lines ahead of the user almost immediately. However, the auto-scroll rate is still usable since speech and gaze balance out motion. In future iterations, it would be possible to add a separate way of visually indicating where the system thinks the user is in order to aid with debugging and to give more transparency for the user.

User Study Recap

Design
We conducted our user studies with 6 people in room 13-1143 across three different days. The first two user studies were used as pilot tests.

In general, we stayed true to the written script that we handed in earlier. During pilot tests, we noticed a few behaviors we didn't expect and incorporated the following modifications afterwards:

- The facilitator gave the user detailed guidance about where to position themselves in front of the laptop during motion calibration. To double check this, she asked the user to play a few different chords to see if the user's left hand was placed correctly.
- The "Viva La Vida" chords did not include the instrumental portions between verses, even though users may play these instrumental chord progressions voluntarily. If the user played these extra chords during the first run of "Viva La Vida", the facilitator would kindly ask the user to play exactly what is written on the page.

Overall, our study went well in that our users understood our verbal instructions and were able to use our system in such a way that led to relevant feedback. Also, we realized that speech detection was not very strong- if we had the chance to do this again, we would have asked our users to use a microphone while singing "Viva La Vida" so that the system could have a greater chance of catching the user's speech.

In addition, we would have either added the instrumental portions of "Viva La Vida" to the lyrics HTML, or explicitly asked every user to play the chords exactly as written. One of our users was not consistent with their playing, so their first run through "Viva La Vida" did not include the instrumentals while the second run through did.

We also would have explicitly told the user that calibration is correct only when the white boxes appear on the webcam output during chord changes. We weren't explicit about what we were looking for in calibration. If we were transparent about this, our user studies could have been more accurate.

Results

The first two pilot studies revealed major issues with the parameters we had set earlier. These incorrect parameters made the system very difficult, if not impossible, to use. The first user had bad experiences with both the static HTML and the MusicBuddy interface. The second user refused to scroll on the MusicBuddy interface and played instrumentals to compensate for lack of scrolling. Thus, the results and feedback from our pilot studies were not as relevant or useful had the system worked at our expected level of performance. However, after we adjusted parameters for the room used for user-testing the MusicBuddy interface decreased the number of interruptions.

After our pilot studies, we updated our parameters which ended up improving overall performance and led to more relevant user study results.

For the user study, we compared the MusicBuddy autoscrolling interface with a static web page, and we counted the number of interruptions. Each user played and sang the song "Viva la Vida" first with the static web page and then with the MusicBuddy interface. Table 1 below shows the results of user testing.

	Static HTML	**With MusicBuddy**
User 1 [Pilot Test]	4	3
User 2 [Pilot Test]	3	0
User 3	3	0
User 4	1	0
User 5	3	0
User 6	2	1

Table 2. Is autoscrolling less interruptive than a static page? Number of interruptions for each user in the user study for the song "Viva la Vida."

For Users 5 and 6, they scrolled an additional time during the Static HTML run because they did not realize they were at the end of the song (they did not need to scroll any more). This is included in the counts recorded in the table above.

Additionally, we took detailed notes on the other observations during the user study because the number of interruptions does not necessarily directly correlate with the user's experience with the system. For example, User 2 did not take their hand off the ukulele to fix the MusicBuddy

system even when the system was incorrect. Instead, they just continued playing instrumentals until the system caught up with their playing. They seemed unwilling to scroll on the interface.

The following table contains relevant qualitative observations made during these user tests:

	Static HTML	With MusicBuddy
User 1 [Pilot Test]	● Played instrumentals between verses and choruses ● Adjusted seat multiple times during runthrough ● Soft voice and soft strumming	● System started scrolling upwards- user tried to correct by scrolling down but system dominated ● Soft voice and soft strumming
User 2 [Pilot Test]	● Scrolled too fast (not used to Mac scroll sensitivity) so needed to 're-scroll' after scrolling ● Soft voice and soft strumming	● Even though system was off, continued to sing because user already knew the words to the song ● When system lagged a lot, user vamped on four chords until the system caught up to where user was ● Soft voice and soft strumming
User 3	● Played instrumentals between verses and choruses ● Overall soft voice and strumming	● Overall pretty soft voice and strumming
User 4	● Ukulele positioned more at an angle than horizontal ● Played louder and slower	● Ukulele positioned more at an angle than horizontal ● Played louder and slower ● System scrolled too fast (current lyric line was then off the screen), so user restarted in the middle of song after pausing for a bit
User 5	● Tried with static page twice because forgot how song went	● Played more slowly than others ● Facilitator sang the words

	● Facilitator sang the words with user during the second runthrough with static HTML	with user
User 6	● Ukulele positioned more at an angle than horizontal ● Played song very fast ● Squinting at screen occasionally to see lyric text	● Ukulele positioned more at an angle than horizontal ● Played song very fast ● System was scrolling slower than the user had wanted, as evident in the need to manually scroll down

Table 3. Qualitative Observations During User Testing

Pilot Testing Insights

Our system's parameters are very dependent on the lighting of the room and laptop/ukulele placement. For example, gaze was unexpectedly more reliable in our testing room than in our dorm rooms, which was where we had tested gaze functionality originally.

Because of this, we decided to give more weight to the gaze modality in our fusion. Also, we increased the motion detection engine's sensitivity so that the motion would always be slightly more ahead in the song than the user actually is. Because the user's gaze is generally in the middle of the page, these two modalities fused together yield decent performance.

Also, speech recognition was given less weight in our fusion step because users (including ourselves) needed to be very loud in order for the system to recognize our words, and the system does not recognize the precise lyric line very accurately.

In terms of ukulele and laptop positioning, we realized that the table-chair height difference influences the system's performance. Because the testing room's table and chair were different from that which we tested with individually, we updated the parameters to address this.

In addition to measuring interruptions, we gave users a survey to measure usability. After the two pilot tests, all of the remaining four users agreed that MusicBuddy was intuitive and easy to use, automated scrolling was helpful, and they would prefer playing ukulele with the MusicBuddy interface over a standard chord website. The table below describes the results of our Likert test. In general, users found our system to be useful.

	Intuitive and Easy to Use?	Automated Scrolling Helpful?	Calibration Annoying?	Prefer MusicBuddy over standard chord sites?
Strongly disagree	0	1	1	1
Disagree	0	0	2	0
Neutral	1	1	2	1
Agree	3	2	1	3
Strongly Agree	2	2	0	1

Table 4. Likert test results from 6 user tests. Green highlighting indicates positive user feedback.

Overall User Testing Insights

Ukulele fretboard position is important during calibration. From our observation, users who held the ukulele at more of an angle had less enjoyable experiences with our system than those who held the ukulele at more of a horizontal position. If the user needs to adjust their ukulele to an unnatural position during calibration, chances are that they will revert to their natural position when playing. Therefore, our method of calibration may not actually make sure the user is calibrated to our system (and our system certainly does not calibrate to the user at this time).

Based on our user testing, we discovered three key results:

- Calibration is very crucial to our system's performance. Because our system does not change its parameters to fit different types of ukulele players, we may need to adjust our calibration process.
- Different rooms, positions, lighting, posture, etc affect parameters and system performance greatly as well.
- In fused scrolling, voice is least helpful, and even though it's noisy, gaze is helpful.

Future Work

As our MusicBuddy currently stands, the user must adapt to the system rather than the system adapting to the user. Given how sensitive our system is to different variables such as room lighting, the usability and flexibility of our system would vastly improve if it could automatically adjust its parameters based on its environment. For example, the system could adjust the weight

it applies to gaze tracking and motion detection based on the brightness of the webcam input. For brighter rooms, motion detection would be made less sensitive and gaze tracking would be given greater weight. For darker rooms, motion detection would be made more sensitive while gaze tracking would be given less weight.

Another minor improvement would be to have a normal webcam output for motion detection, rather than the default output provided by the diff-cam-engine library. It is not very obvious that it actually is output from a webcam and thus is not intuitive for the user to move their hand into the motion detection box in the lower right corner.

Alternatively, we could also go with a much lower-tech solution by using a multi-column view for chord charts. You can see Elaine's solution here.

Work Distribution

All members of the team worked on deciding what technologies to use, how to fuse modalities, and how to scroll. Becky refactored the code and added a toggle feature for all of the modalities. Phoebe implemented the fusion code and set the weights for the modalities. Elaine added Mustache templating for songs.

Tools

We used libraries for tracking gaze, speech, and motion, and our impressions of the tools changed as we polished our interface. Originally, scrolling was more heavily dependent on speech and motion. After parameter testing, scrolling was more dependent on gaze and motion.

The Webgazer library is not that precise, but useful for estimating the user's location. Increasing the amount of the light in the room improved accuracy and decreased lag.

Although the WebSpeech API performs well when tested alone without a ukulele, most users sang too quietly for the system to detect any speech. In addition, the naive matching algorithm for speech (checking if the user says two or more words on the next line) does not work if the system is slightly off in estimating the user's location in the song.

The diff-cam-engine library is also not precise by itself. We needed to modify the library so that it would only focus on the user's left hand on the fretboard. Scrolling using both gaze and motion was better than scrolling using only motion.

Performance also depends on the angle of the ukulele neck and the position for holding a ukulele. As shown in the user testing, performance suffered when users held the ukulele neck too vertically.

References

- Webgazer
- WebSpeech API
- diff-cam-engine
- Mustache templating
- MusicBuddy github
- Demo site