Dance Augmenter

Andy Kuang, Priscilla Wong

Abstract

The Dance Augmenter allows for dancers to use a combination of gesture and speech commands to control Youtube video settings from a distance. There are six functions users can perform on the video (pause, play, rewind, fast forward, speed up, and slow down), each with its own mapping to a pose and a voice command. Our system uses the Kinect sensor with SimpleOpenNI to track body movements and the laptop microphone with a speech recognition library to process voice input. It worked quite well for pose detection when the user is about 10 feet away from the Kinect and was accurate in speech recognition when the user is close to the microphone. Due to how the SimpleOpenNI determined the location of joints and our joint relation constraints, the false negative rate was higher than the false positive rate, except in the case of the resting pose. Once a command is detected, the simulated keypresses are instantaneous and result in an immediate change to the video settings when Youtube is in focus.

Introduction

Let's say that Alice is a dancer who is learning a piece of choreography from a Youtube video for an upcoming performance. However, the music is fast and the choreography is difficult. As Alice is learning, she repeats different sections again and again, starting from a slower video speed and incrementally getting faster. Every time she wants to rehearse a particular section of the song or change the speed, she has to stop her dancing, walk over to her laptop sitting at the front of the studio, manually make the changes on Youtube, and then back away from her laptop so that she can dance comfortably and see her full body in the mirror. What if the video and audio naturally pauses when she stops dancing and she can control the Youtube video from a distance, with speech commands such as “go back 10 seconds” or “speed up” and gesture commands such as standing in a resting position to stop the music or raising a hand to resume?

When practicing a piece, dancers often walk back and forth between their laptop and the center of the studio just as much as they dance to the music. We envisioned the Dance Augmenter to enable more suitable interactions between dancers and their video/audio source using gesture and speech recognition to reduce unneeded mechanical movement so dancers can focus on the choreography and get the most out of their practice time.

System Overview

The Dance Augmenter uses the Kinect to allow users to make various voice and gesture commands to control the settings of a Youtube video. Using SimpleOpenNI, we connected to the Kinect in Processing, an integrated development environment, and programmed our system to recognize specific poses based on joint relations. Using a Speech Recognition Library for Processing, voice input from the laptop microphone was transcribed into words and this was used to determine whether specific commands were said. Based on the inputs, the system simulates the corresponding keypresses to change the video settings in Youtube. A typical user flow would be to pause the video, rewind a little bit, slow down the speed, and resume. A user can do this with the Dance Augmenter by assuming a resting pose, saying "rewind 10 seconds", saying "slow down", and raising his/her right hand.

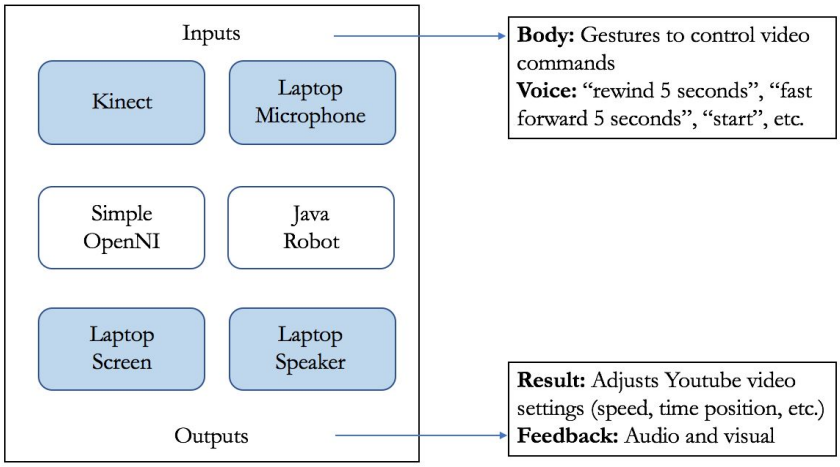

Figure 1: System Diagram

Implementation

State Machine

We modeled our system as a state machine with various states and state transitions. A state machine diagram is shown in *Figure 2*. The system begins in the START state, during which the system is actively looking for the user to assume a resting position. Once the resting position is registered, the Youtube video is stopped, and the system enters the TRACKING state.

In the TRACKING state, the system is actively looking for the supported gestures and voice commands (rewind, fast forward, speed up, slow down, start). Once a supported gesture or voice command (with the exception of start) is detected, the system adjusts the Youtube video setting appropriately, and remains in the TRACKING state.

Once a "start" gesture or voice command is detected, the system starts the Youtube video and moves to a REFRACTORY state. This REFRACTORY state ensures that the system does not start searching for commands immediately after the video started, avoiding situations for which the video is stopped immediately after it starts. After staying in the REFRACTORY period for 5 seconds, the system moves back to the START state.

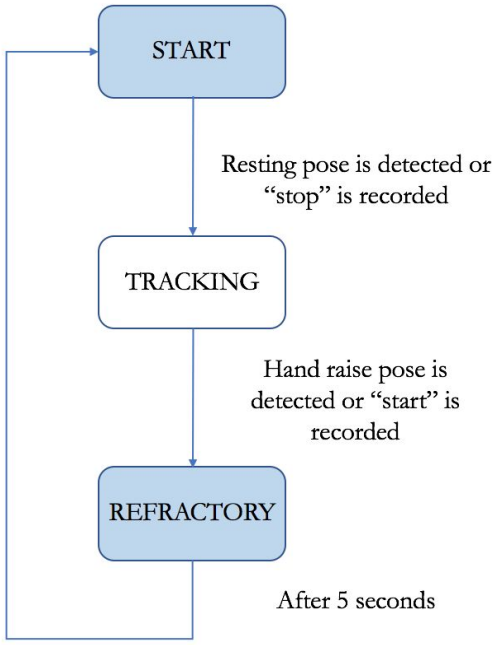

Figure 2: System State Machine

Pose Detection

To perform pose detection, we modified a given SimpleOpenNI example called SkeletonPoser. This takes a Kinect feed, finds and identifies the joints of the user in view, and overlays a skeleton, as seen in *Figure 3*. We define a pose as a series of rules constraining joint relations such as relative x-y location, distance, and vertical/horizontal alignment. The skeleton of the Kinect feed is checked against the constraints of all specified poses, and if it satisfies the rules of a pose, the appropriate action is triggered.

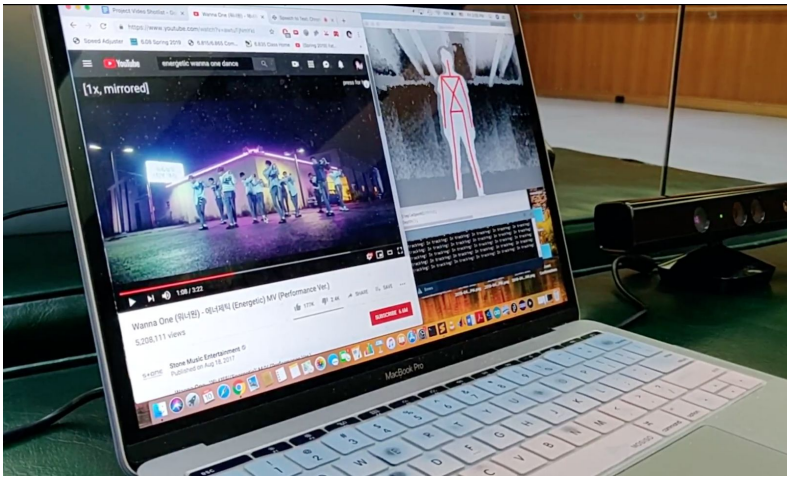

Figure 3: Kinect Skeleton Tracking Feed

We decided on these poses based on intuitiveness, simplicity, and distinctiveness. The 6 poses supported by our system and their functions can be seen below in *Figure 4*.

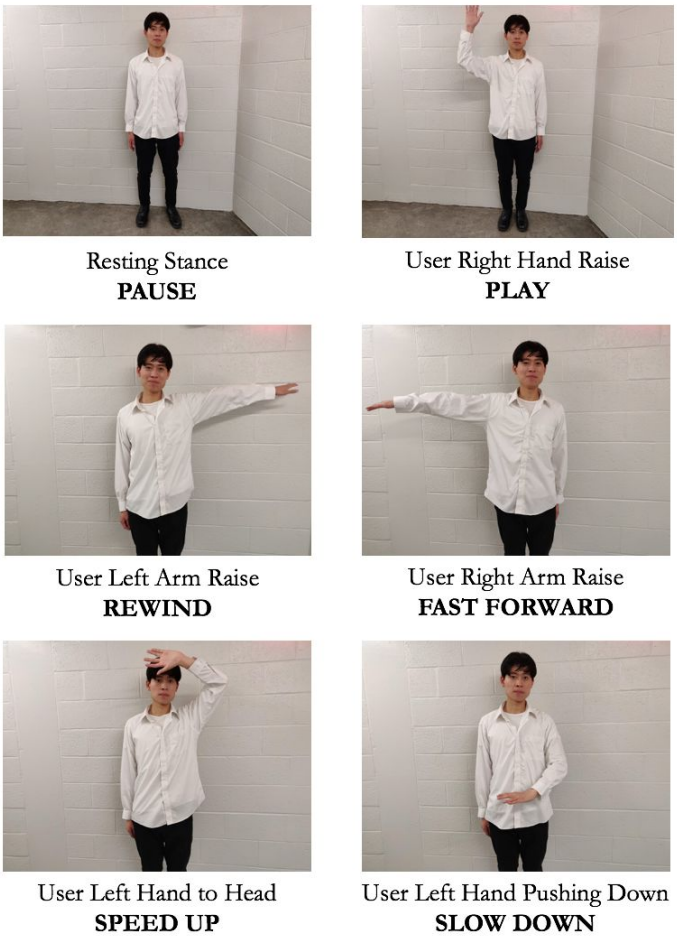

Figure 4: Pose Commands from the Kinect Point of View

Speech Recognition

Our system also supports speech input. We implemented speech recognition through the use of a Speech Recognition library written by Florian Schulz. The speech recognition process works as follows - we open a new WebSocket in our Processing sketch at a specified port (1337). Then, an accompanying Codepen application (open in Chrome) listens for incoming voice input. The text transcript of the voice input is then sent to our Processing sketch via the open WebSocket. At this point, if the crucial key phrases are in the text transcript (such as "start" or "stop"), then the system will perform the corresponding action on the Youtube video.

Below are the voice commands supported by our system and their functions:

Voice Command	Youtube Function
Stop	Pause the video
Start	Resume the video

Rewind	Rewind 5 seconds
Rewind **X** seconds	Rewind **X** seconds
Fast forward	Fast forward 5 seconds
Fast forward **X** seconds	Fast forward **X** seconds
Slow down	Slow down 25%
Speed up	Speed up 25%

Output

Keypresses

The main functionality of the Dance Augmenter is changing the settings of a Youtube video. Our implementation takes advantage of built-in Youtube keyboard shortcuts by simulating the corresponding keypress to control the video given an input speech or gesture command. To do this, we use the Java Robot class, which takes control of the keyboard and can perform a specified stroke. This requires that the window of focus is the Youtube video being played. Below are the relevant mappings of keypress to function:

Keyboard shortcut	**Youtube Function**
Space	Play/Pause the video
Left arrow	Rewind 5 seconds
Right arrow	Fast forward 5 seconds
Shift + .	Slow down 25%
Shift + ,	Speed up 25%

Speech

For better response of system status, we also implemented speech output in response to the user's actions. For example, the system would respond "video rewinded X seconds" or "video sped up," after the user performs the corresponding actions. This is extremely helpful especially for the rewind, fast forward, speed up, and slow down operations, since from a distance, it is difficult to see if a particular video frame has changed, especially if the video frames don't change much.

Speech output is implemented by leveraging the power of Mac's built-in text-to-speech engine, used primarily in accessibility contexts. To invoke the engine, we call the *say* command, followed by the desired speech to output. To do this via the Processing sketch, the system creates a new Java Runtime Execution object spun up in a separate thread, which handles sending the say command to the Terminal shell. Once the

say command is handled, the thread and the object are both terminated, ending that particular speech output instance.

User Interface

After conducting our user study, we designed a user interface that maximizes video space and provides feedback with the most relevant system information. When the Kinect loses track of the user, the "USER LOST" screen is displayed**.** Otherwise if the Kinect is tracking the user and the system is in the START state, the "VIDEO PLAYING…" screen is displayed, indicating that the system is expecting the user to be dancing and is actively looking for the rest position. Once the system registers a rest position, a red border is displayed to indicate that the video has stopped, and a speed tracker is included in the upper right corner of the screen to inform the user of the current video speed. Then after a gesture is performed, the result of the gesture input is displayed for two seconds, also updating the speed tracker if it was a speed adjustment. Upon starting the video again, the "VIDEO PLAYING…" screen is displayed once again.

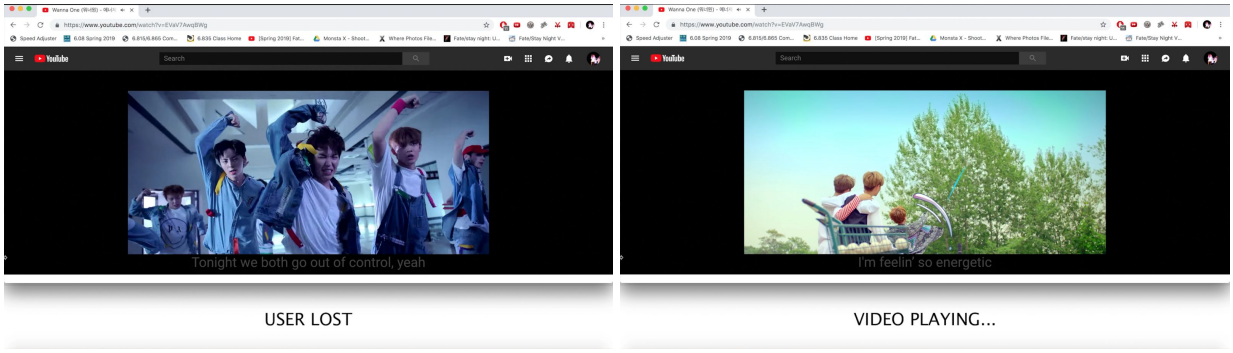

Figure 5: User Interface (user lost and video playing)

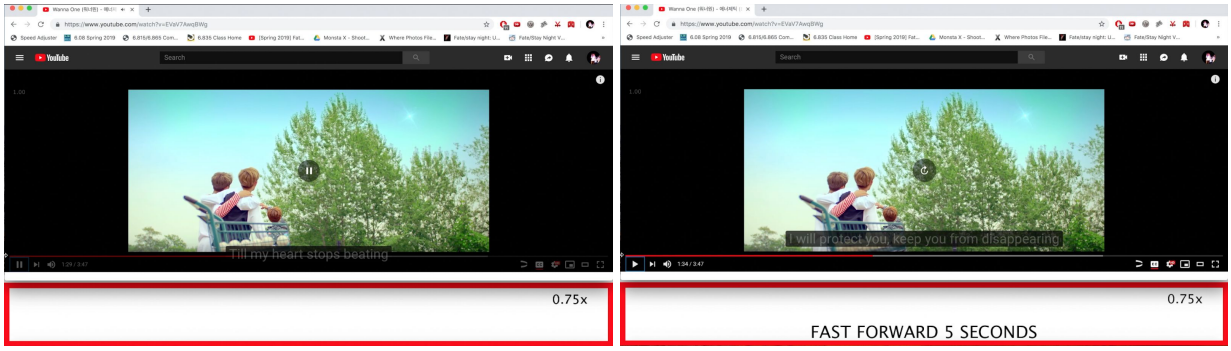

Figure 6: User Interface (tracking and command detected)

User Study

We designed our user study to be a formative evaluation on the usability of our system. We carried out the user study with a complete design and asked the users to first learn a dance as they normally would and then learn a dance using our Dance Augmenter. During testing, we observed the users' behavior and noted when they struggled or the system made an error. The users also compared the two experiences, graded

the experience using our system, and provided feedback to improve usability based on their experiences. Below is the design that we prepared:

Part 1: Welcome and Introduction (2 min)

1. Welcome the participants.
2. Ask them if it's okay that we are filming them and taking photos.
3. Introduce our project.

Part 2: Control condition - Normal learning (5 min)

1. Have the participants choose a 15 second section of an intermediate dance that the participants have never learned.
    a. Boy With Luv (1:00 - 1:15) https://www.youtube.com/watch?v=CzvfbRbEjww
    b. Fancy (1:05-1:20) https://www.youtube.com/watch?v=iRw4kL1CMI8
    c. I'm OK (1:05-120) https://www.youtube.com/watch?v=VGwySDLllPc
2. Have the participants start learning the dance as they normally would.
3. Observe and record how many times they stop and start the video and walk back and forth from their laptop.

Part 3: Training (5 min)

1. Teach them the basic gestural commands.
2. Teach them the various voice commands.
3. Demonstrate example use.

Part 4: Testing condition - Learning with the Dance Augmenter (10 min)

1. Have the participants continue learning the dance with the Dance Augmenter.
2. Observe and record any (and type of) recognition failures, any (and type of) user input failures, and overall user satisfaction.

Part 5: Debrief (2 min)

1. Ask participants for general feedback and comments.
2. Ask participants these questions:
    - How difficult on a scale of 1-5 (1 easy and 5 difficult) was it to use the system?
    - Would you use this as a tool in learning dance? If not, what improvements can be made to make it more effective?

We had two participants in the user study, both of whom were MIT Asian Dance Team choreographers who often learn dances from Youtube videos. When learning the dance normally, we found that both participants started out right in front of the laptop concentrating on the video in order to understand the moves and frequently rewind and play. They only started using the mirror after reviewing the choreography enough times to be comfortable with the individual steps, suggesting that our tool is more likely useful after a certain level of familiarity with the dance. *Figure 7* depicts the typical setup of the room in order to use our system, which is followed during the testing condition of the user study.

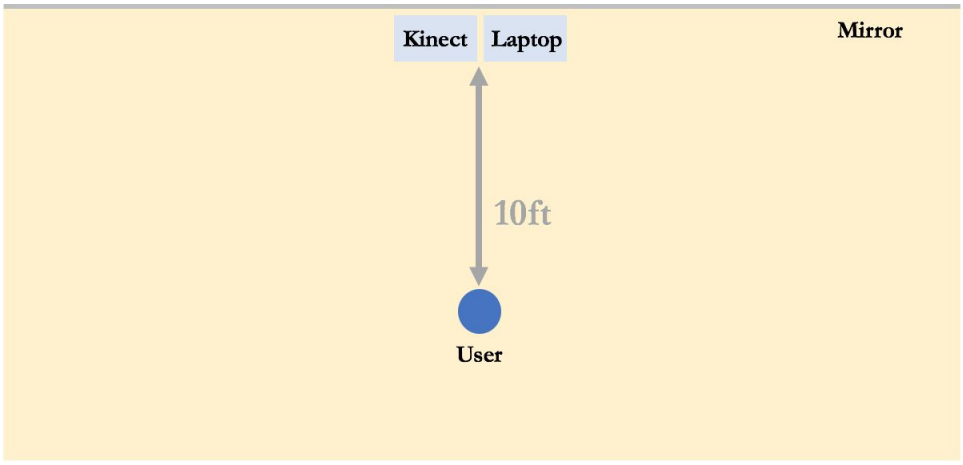

Figure 7: Setup for Use

The users described that it was fairly intuitive to learn our chosen commands. We observed that a common command pattern was to stop, rewind, change speed, then start, leaving fast forward rarely used. During use, the Kinect lost track of the users several times, displaying a limitation of the technology. However, when this happened, the users relied on the visual feedback of the skeleton tracking feed to recognize the failure and moved around to get tracked again by the Kinect. With regards to system feedback, the users found it useful to have audio announcements when commands were successful and visual feedback on whether the Kinect was tracking them or whether their poses were being correctly detected. However, the laptop screen has limited real estate, and users strongly expressed the need for the Youtube video to be larger, valuing this over any space-intensive visual feedback. The system also had difficulty distinguishing intentional stop commands vs. unintentional stop commands given that the stop command was a resting pose. Occasionally this improved usability because when the users were confused about a part of the dance, they naturally assumed a resting pose and the video stopped before they were even conscious of their actions. However, on the other hand, the system would also see the user assuming a resting pose while trying to learn the dance. This happened around 15 times for each user in the testing condition and occurred more frequently when the dance was at a reduced speed.

Although usability improvements would be necessary to make our Dance Augmenter preferable to physically controlling the video and more effective as a tool in learning dance, the users expressed excitement about the overall idea of our project. After the user study, we completely changed the visual feedback of our system and designed a simple user interface. In accordance with our observations and the participants' comments, the new user interface maximizes space on the screen for the video and displays only the most useful information like tracking state, system state, and action performed.

Performance

Overview

In general, our system works pretty well for pose detection. Due to the limitations of the SimpleOpenNi library as well as the joint relations that we specified, there are many cases of false negatives when the user does not perform a pose as expected by the system. An example of this was the rewind

operation, where the user does not stick out the left arm horizontally enough to count as a rewind pose (even with the tolerance for variation implemented in system). However, there were almost no false positives in the TRACKING state, since the simplicity and distinctiveness of the poses makes it rare for one pose to be mistaken for another. But in regards to usability, our system works well for pose detection in the TRACKING state, since the user quickly learns what an accepted pose of our system is and can adjust accordingly.

On the other hand, our system does not perform as well for detecting the resting position, due to challenge of eliminating false positives during dance. There were many instances in which unintentional stops occur, such as shortly after a jump, which we were unable to completely address in our design.

The performance of our system for voice input, however, was highly dependent on the acoustic environment. In a room with great acoustics, the voice input was detected well, without requiring the user to speak unnaturally slowly or loudly. However in an echoey environment, the voice input from the user is not crisp, which causes it to not be picked up well by the system. In this environment, the voice input only sometimes works if the user speaks unnaturally slowly or loudly.

One very interesting failure that we encountered early on was the scenario in which voice output was fed back as voice input. This resulted in commands being executed twice, since the initial design of the rewind operation only looked for the word "rewind," which the speech output ("video rewinded 5 seconds") contained. To address this issue, we had to change the implementation to exclude transcripts that contained the word "video," as that was a flag denoting system output. This was an interesting failure, because we learned that this was a common challenge faced by any multimodal user interface involving voice (such as Siri, and even other classmates' final projects), which taps into decades of research that exist in this field.

Limitations

Sensitivity of Voice Input to the Environment

A major limitation that our system has is that the success of voice input is heavily sensitive to the environment. When testing the system in a room with good acoustics, voice recognition works pretty well even if the user is farther from the laptop. However, when testing the system in a particularly echoey dance studio, our system had difficulty registering our voice input from a distance. In these extremely echoey locations, voice input was not detected very well, and when the system did successfully detect the voice input, it was for cases when we spoke very loudly (much louder than a comfortable speaking volume). To test the feasability of voice input as another modality, in the future, we would test this system in many more dance studios, rooms, and general spaces with mirrors, to see if the environment renders this as effective. Currently, the unreliability of voice input decreases the usability of our system.

Refractory Periods for Voice Commands

Another limitation that we have in our system currently is that there is a refractory period after each voice command. This refractory period is necessary in our system, because despite trying many different approaches, we are unable to handle the situation in which a user quickly gives a second voice command after the first voice command, interrupting the voice output from the system and causing a long, bogus input to the system. For example, if the user says "rewind 15 seconds", followed by "fast forward", the possible voice

inputs to the system could be "video rewinded 15 seconds fast forward," or "video rewinded fast forward", or other combinations of the like. We tried to implement parsing logic to handle the quick second command after the first, but because of the various unpredictable voice inputs caused by the interruptions (in addition to general speech recognition challenges), this task proved to be too difficult. Instead, we decided to have a short refractory period after each user voice command and system output command cycle.

Granularity Constraints

The main limitation that we dealt with stems from the fact that we simulate Youtube keyboard shortcuts to adjust the Youtube video. As a result, we are constrained by the different granularities that the Youtube keyboard shortcuts provide us. For example, the rewind and fast forward commands can only be done in multiples of 5 seconds, and likewise, the speed up and slow down commands can only be done in multiples of 25%.

As a result, if the user provides a certain number of seconds that is not a multiple of five, we decided to divide the number of seconds by five, round the result to the nearest integer, and multiply it by 5 seconds to obtain the number of seconds to rewind or fast forward. Mathematically, if x is the number of seconds that the user says to the system, then the actual number of seconds rewinded, y, is y = 5*round(x/5.0). We believe that given the limitation from the Youtube keyboard shortcuts, this best approximates the number of seconds to rewind or fast forward. To avoid potential confusion, the voice output from the system outputs the actual number of seconds (multiple of 5) that is rewinded or fast forwarded, informing the user.

Implementation Review

Challenges

Kinect and SimpleOpenNI

The initial challenge that we as well as several classmates faced was getting the Kinect to work properly on the Mac. This required significant trial and error with many different libraries, until we found one that worked. However, even after getting the Kinect to properly work on the Mac, it was difficult to figure out how to use the library, since there was a severe lack of documentation. Because we were unable to figure out how certain functions worked, we had to make a major design pivot from our original idea. Originally, we planned to implement moving gestures for rewind, fast forward, speed up, and slow down, as those were more intuitive for the user. However, we were unable to get the included NiTE library (bundled with SimpleOpenNI) to work with moving gestures, since without any documentation, we could not even understand what the raw input was for moving gestures. We therefore decided to scrap the idea of moving gestures and finalized on only static gestures, which we knew was less intuitive but was an accepted drawback given the inability of getting moving gestures to work.

Joint Constraints vs. Usability

One major challenge that we constantly faced was balancing the tradeoffs between joint constraints and usability. For pose detection, we wanted to ensure that constraints were restrictive enough such that

different poses can be easily distinguished by the system, without mistaking one pose for another. However, the other factor that we needed to balance was usability - if poses were too restrictive, then users would be forced to assume a very particular pose in order for the system to register it, which would place additional, unnecessary cognitive load on the user.

The most prominent example of dealing with this tradeoff was with the resting position. Before our prototype demonstration, we specified that a user's legs must be shoulder-length apart. However during the prototype demonstration, we learned that the constraint was too restrictive, as many users who tested our demonstration had a wide range of distances between their legs when they assumed a resting position (in one case, one student even had their legs completely together). We did not anticipate this issue prior to the demonstration, which turned out to be crucial feedback that encouraged and required us to relax our constraints between the legs, while at the same time keeping the resting position distinct from the other poses. We decided to implement a max distance metric between the legs, set at approximately shoulder-width apart, and counted any distance less than or equal to the max distance to count as a resting position. At the same time, we enforced constraints between the knees and the feet, enforcing that the knees are inside the feet. With these series of constraints, we are able to exclude poses and dance moves that are clearly not rest positions (such as squats), while at the same time giving the user a lot of flexibility regarding the distance between the legs.

Unintended Stop Triggers During Transitionary Motion

When people stop dancing, they often assume a resting position. This natural behavior often precedes a person's conscious desire to stop the music and any actions to do so. This was the motivation behind having a resting pose as the stop gesture. Another major challenge that we encountered was the issue of undesirable false positives of the resting pose during transition motion. This happened a lot during dancing - for example, when we assume a quick transitionary resting position during a dance move, the system would instantaneously register that as a resting position and stop the video. To address this issue, we decided to only register a pose if the pose is present for at least 25 frames (determined empirically). This was a design choice we made to reduce the frequency of transitionary poses not being registered, while at the same time not harming usability by ensuring that the user does not have to assume a pose for too long before it is registered.

Accidental Stop Triggers After Start

Another challenge that we encountered was the issue of unintended stops when starting the video. This issue manifested itself immediately after starting a video - if at any point the user drops his/her right hand (to relax from the hand-raised position in order to start dancing), then the system immediately registers that as a resting position, which immediately stops the video. This was an undesirable behavior, because it would impose a major cognitive load on the user to never assume a resting position during the transitionary period from the hand-raise to dancing.

We overcame this issue by imposing a REFRACTORY state in between the TRACKING and START states, with a duration of 5 seconds. During this refractory period, no poses will be tracked. This gives the user more time and comfort in adjusting from the starting position to the dancing position, and prevents any unintended stops of the video.

Voice Output Perceived as Voice Input

One major challenge that we did not anticipate was the issue of voice output being fed back as voice input to the system. Without accounting for this issue, the result was that when we said a command such as "rewind 15 seconds," the voice output of "video rewinded 15 seconds" triggered another rewind operation, causing the video to rewind 30 seconds. To account for this, we excluded processing any text transcript that had the word "video" in it, which properly distinguishes it from user voice input.

Distinguishing Between Different Phrases

Another major challenge was getting the system to recognize certain voice commands. These were issues that we did not consider at all beforehand, and were only apparent during the testing phase. In particular, our system had major issues registering the phrase "speed up," as our system sometimes registered the phrase "beat up" when we said speed up. Since this was consistently the only other phrase that was registered, we overcame this issue by accepting both "speed up" and "beat up" as valid phrases to speed up the video.

Voice Output Pronunciation

A small challenge that we had to deal with was the fact that for certain words, the canonical manner in which a word is pronounced is not accurately reflected in the speech output. For example for the word "rewinded," the system pronounced the "winded" part the same way that "wind" (the breeze) is pronounced, which was quite jarring. In order to get the system to pronounce the word canonically, the text that I input to be spoken was "rewineded," which was an intentional misspelling that produced the desired pronunciation.

Studio Feedback

Studio feedback was helpful in addressing some design challenges that we faced. Since we pivoted from moving gestures to static poses, the challenge was deciding which static poses were the most intuitive for the user. We therefore asked many classmates who tested our system what the most intuitive static poses would be, and our choice of poses for rewind, fast forward, speed up, and slow down was based on this feedback.

We also received suggestions for other functionality that our system can provide, such as allowing users to input their own custom gestures or using machine learning for higher-accuracy pose detection. We weren't able to implement these other suggestions due to the difficulty in scope and lack of time, but these are definitely suggestions that can be incorporated for future iterations of the system.

Future Work

To improve usability of our system, we would need to overcome the granularity limitations imposed by the Youtube keyboard shortcuts. For example, there would be much better usability if a user can rewind or fast forward a specified number of seconds that is not a multiple of five, or jump to a specific point in the video. In order to do so, the implementation of the system would have to change - instead of simulating

keyboard strokes to manipulate the video, we would need to directly manipulate the HTML & Javascript of the webpage to adjust the Youtube video. This would require much more knowledge and experience that is beyond the scope of this course, but is definitely future work that can be done to improve the system. In terms of difficulty, this would require comfort and expertise in web programming at a native level, requiring directly manipulating elements without the aid of outside packages and libraries.

Furthermore, with both speech and gesture recognition, minimizing false positives and false negatives would be necessary to increase usability. For speech, this requires an improvement of recognition given distance or noise, because we had difficulties when the user was far from the microphone or in an echoey room. These improvements, however, would be difficult for us to implement given the limits of the best speech recognition software today. For gestures, using machine learning rather than joint relations would allow for less constrained recognition of poses and make the system more adaptable to the user. We would definitely be able to take this approach to perform gesture recognition but we would need a stronger background in machine learning to reach a high accuracy rate.

Conclusion

The Dance Augmenter aims to eliminate the unnecessary mechanical back-and-forth movement present in the current process of learning choreography by allowing dancers to use various speech and gesture commands to control Youtube video settings from a distance. Because of this project, we were able to not only develop a good proof of concept of a cool, viable application of multiple modalities but also recognize the limitations brought to our attentions of these modes of input and output. With knowledge of current technology and approaches, we learned to think more critically about the interactions between user and product and reimagine the possibilities in the world around us.

Author Contributions

Dance Augmenter was designed and developed in a collaborative effort by Andy Kuang and Priscilla Wong. Component testing and the user study were conducted together. The overall deliverables contained equal contributions from each author.

Resources

Processing 3.5.3

https://processing.org/download/

The code for our system utilizes Processing, for both the SimpleOpenNI Kinect code as well as the user interface. Our system was tested on Processing 3.5.3.

Simple OpenNI for Processing

https://github.com/totovr/SimpleOpenNI

The author of this library was able to get the Kinect to work with the Mac, using Processing. The included demo code (depth maps, body tracking, joint detection) worked very well and was a good springboard to our main project. In addition to cloning this library, we followed the instructions under the instructions.md file to install libfreenect, as well as to copy the file `libfreenect2-openni2.0.dylib` to the Processing Simple OpenNI library folder. The library tested to work on both Mac High Sierra and Mac El Capitan.

Making Things See by Greg Borenstein

https://www.oreilly.com/library/view/making-things-see/9781449321918/
There is a 10-day trial period on this website to view *Making Things See* for free. Although a bit outdated, it was still a useful book documenting ways to use the SimpleOpenNI library. We consulted the book to understand the library and implement pose detection.

Speech to Text Recognition Codepen for Processing - by Florian Schulz

https://codepen.io/getflourish/pen/NpBGqe
Prior to using the Codepen, follow the instructions on http://florianschulz.info/stt/ to install the WebSockets library from https://github.com/alexandrainst/processing_websockets and create a Processing sketch that interacts with the Codepen. This is a Speech Recognition library that basically worked out of the box with Processing. A limitation, however, is that everytime the Processing sketch is recompiled, the Codepen must be refreshed.