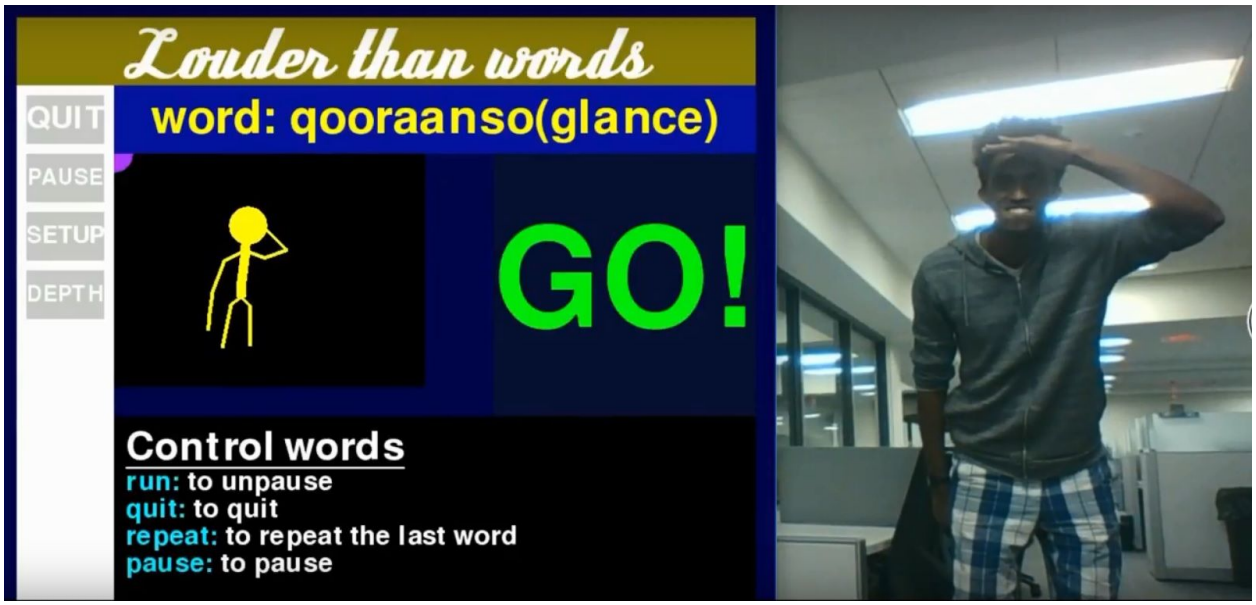
Louder than Words

6.835 Final Project Report

Anelise Newman and Mubarik Mohamoud



Louder than Words is a gesture based vocabulary learning system that lets students learn foreign language vocabulary in a mentally and physically engaging way. Students learn a mapping between vocabulary terms in a foreign language and an intuitive gesture that is emblematic of that word. In Practice mode, students get to see both the term and its translation and practice the gesture that goes along with that word. In Testing mode, students only see the original term and have to perform the corresponding gesture. The system then gives them feedback as to whether they did the right gesture, letting students quiz themselves on what they have learned. Gestures are recorded using a Kinect sensor and users can control the flow of the system's UI using traditional mouse/keyboard input or voice commands.

When tested using a vocab set of around 10 words on users who each practiced the system 3 times, our system achieved around 70% accuracy. We also received positive user feedback saying that they enjoyed using the system and felt that they learned the words they were asked to practice.

1. Motivation

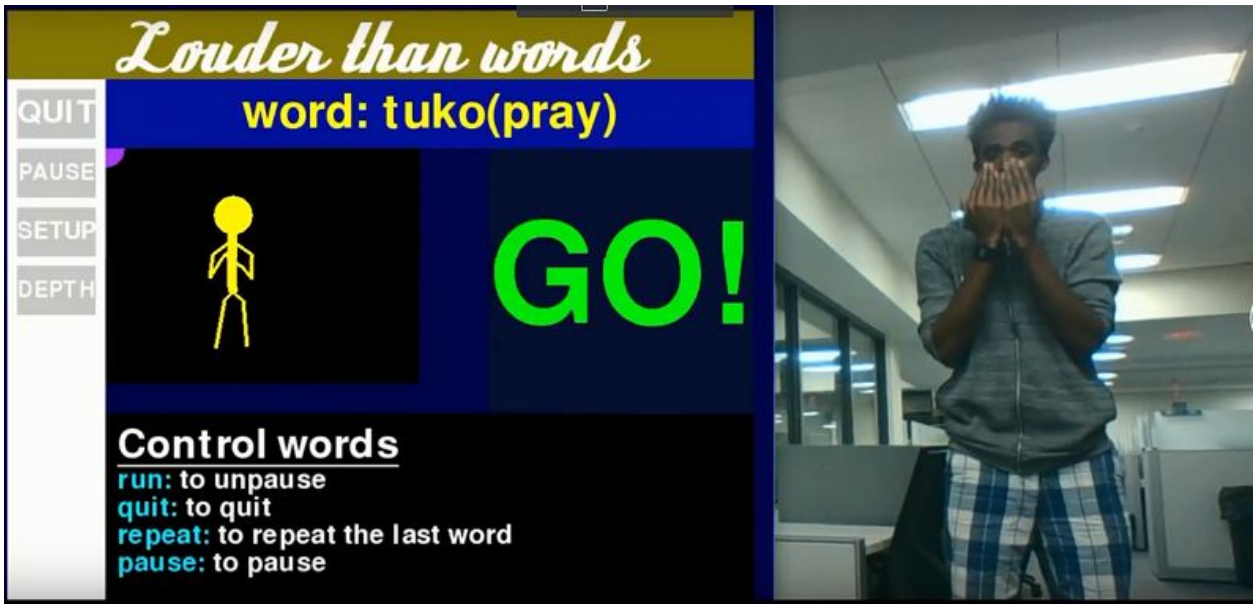
While learning vocabulary in a new language, there is evidence that performing gestures while speaking a new word is more effective at cementing that term in memory than just memorizing the word from a list or even associating it with a picture. Learning via gesture is also beneficial because it associates a term with an idea or action, instead of a translation in the learner's native language.

The goal of Louder than Words is to use gesture recognition technology to create a vocabulary-learning application for students that leads to more efficient and long-term retention of new terms. Students can practice their vocab by performing gestures that are emblematic of that term, and when they're ready to quiz themselves Louder than Words can give them feedback as to whether they performed the right word or not. Louder than Words encourages students to develop a more conceptual understanding of vocab terms by memorizing a mapping from word to gesture instead of word to translation. It also provides a fun way for students to study that's a lot more engaging than memorizing a list of vocab words.

2. System Description

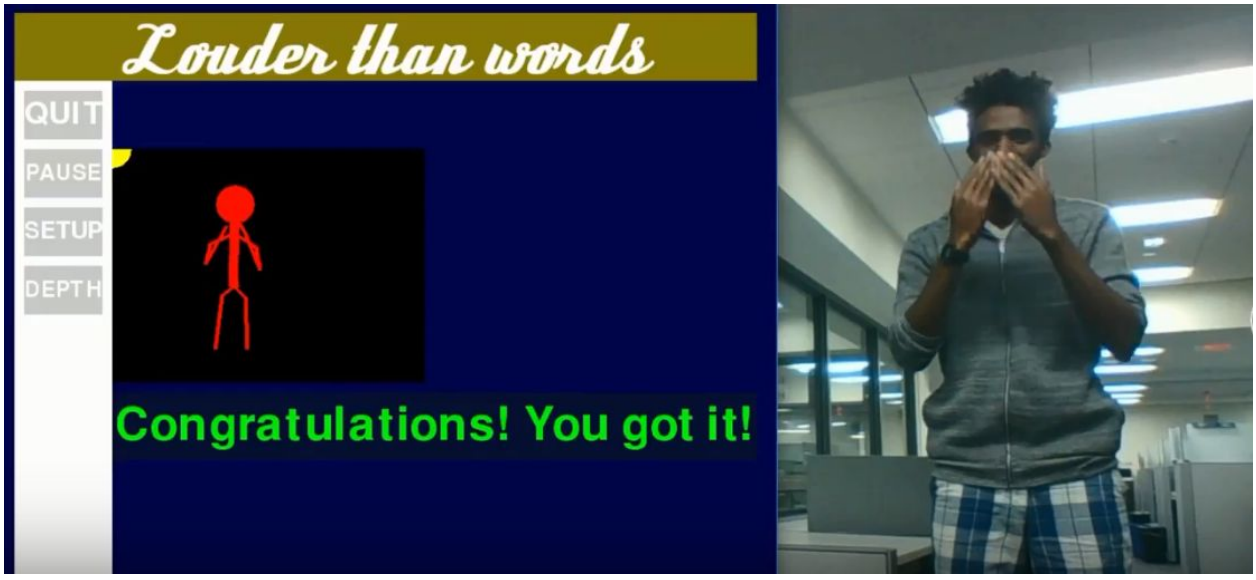
There are two modes to our system: practice and testing mode.

During practice mode, a student is shown both a vocab word in another language and its English translation. They practice performing a gesture of their choice that corresponds to the given word. The student is free to choose their own gesture, which means they can select a motion that is intuitive and easy to remember. During this time, the student is getting familiar with the vocab set, getting exposure to the words, and getting some practice performing their chosen gestures. At the same time, the system is recording the gestures and storing them for future use. We aim to have about three examples per gesture for the system to work well.

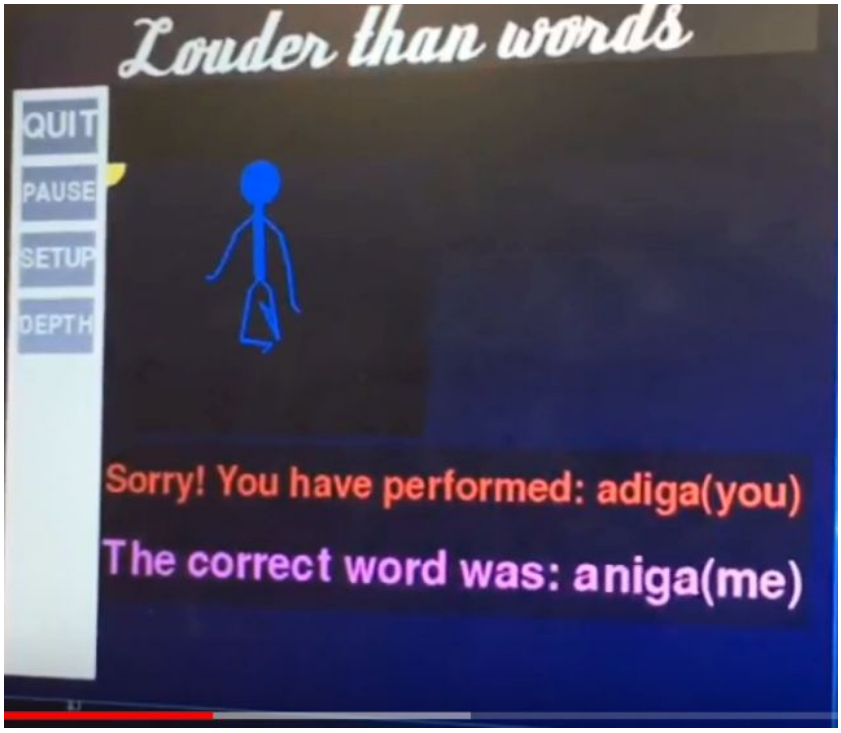


Practice mode

During training mode, the student is shown just the vocab word, minus the translation. They have to remember the meaning of word and perform the appropriate gesture. The system records the gesture they perform, compares it to the training data it recorded in practice mode and gives them feedback as to whether they got the gesture right or not.



Testing mode: positive feedback



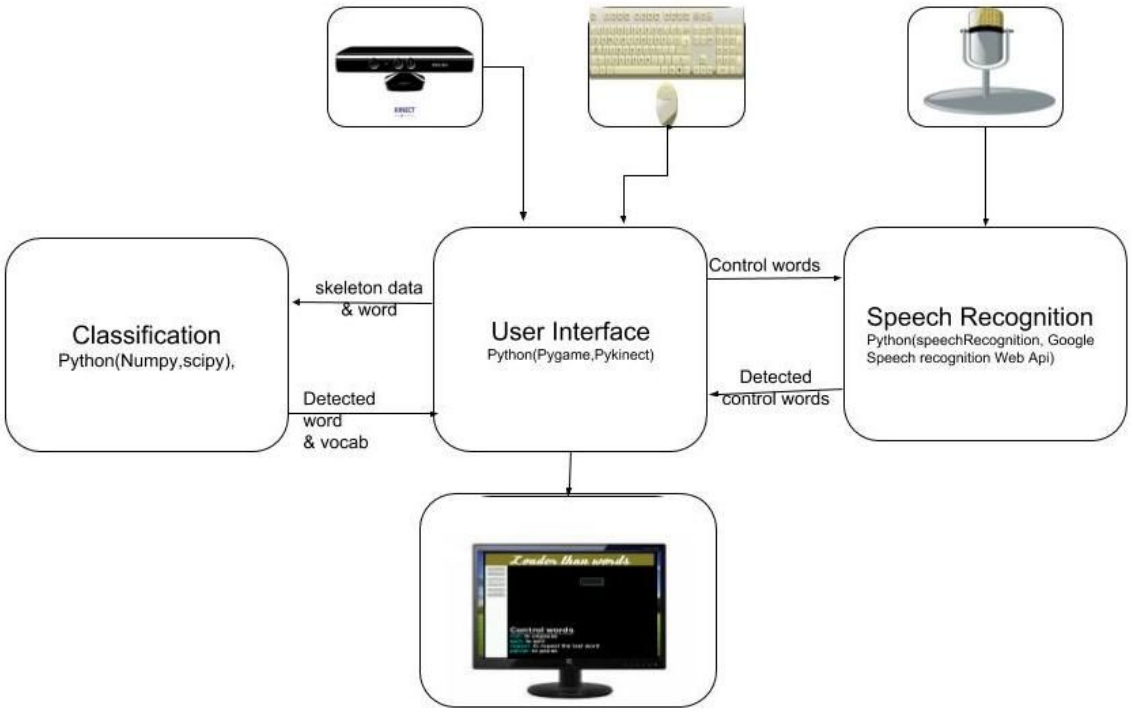
Testing mode: negative feedback

If the student gets the gesture wrong, they get feedback as to both the gesture they performed and the gesture they were asked to perform so they can continue to learn and improve.

Users can navigate the control flow of Louder than Words using both traditional mouse input and speech recognition. The system recognizes the following commands:

- **Train:** Begin practice/training mode
- **Test:** Begin testing mode
- **Pause:** pause the system
- **Run:** resume the system
- **Repeat:** during testing mode, repeat the last word that was given. This might come in useful if a user was not ready to begin practicing or if they get the word wrong and want to try again.

2.1. System Design



A simple system diagram. Our system takes in user input via the Kinect sensor, traditional keyboard/mouse input, and a microphone. Speech is processed using the Google speech recognition web API. Our User Interface code runs the main practice/testing loop of the system, displays the UI to the user, and processes user input. When it needs to make a gesture classification, it sends Kinect skeleton data to our back-end, which implements a nearest-neighbor algorithm in python and returns the predicted gesture.

2.2. Libraries and Tools

Here is a brief overview of the tools we used to build our system:

1. User Interface
    a. Pygame: A python game engine that we have used for displaying UI components, event handling, and timing.
    b. PyKinect: The python wrapper for the Kinect SDK used to read RGB, Depth, and skeleton data from The Kinect sensor.
2. Speech
    a. Speech Recognition: A python speech recognition library that supports most of the major speech recognition engines including the Google Cloud Speech API.
    b. Google Web Speech API: The backend engine used for speech recognition. This backend was used because it worked well and it doesn't require access tokens to use.
3. Backend

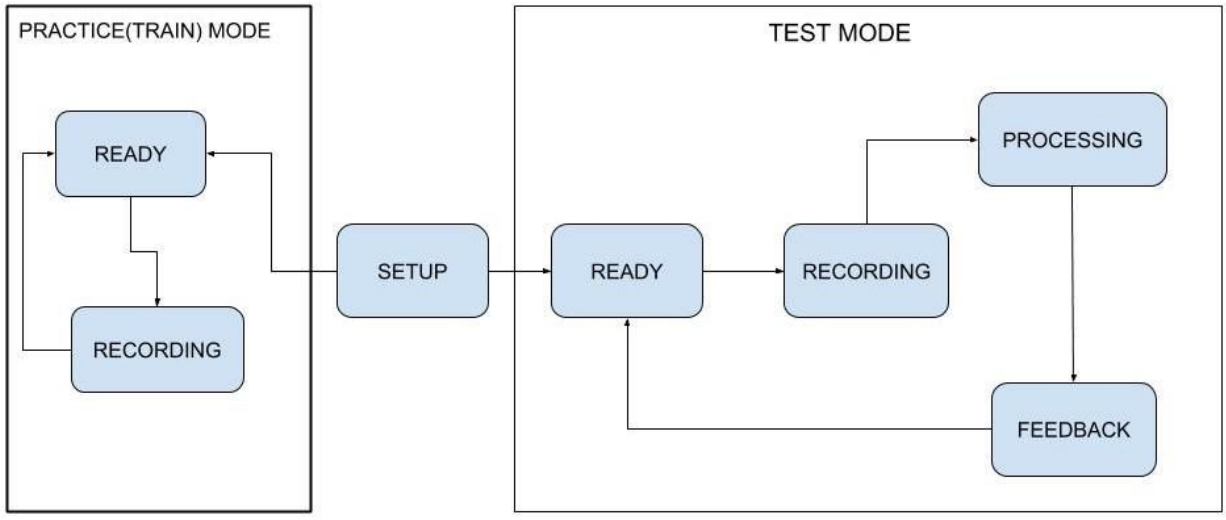
a. For gesture classification, the back-end uses a normalized nearest-neighbor classifier that makes use of the numpy and scipy libraries. We also tried doing classification based on scipy's Decision Tree framework and a neural network built in Keras (not currently in use, although the code is still in place).
b. For our data model, we built upon the Gesture/Sequence/Frame classes provided in MiniProject2. For data storage and access, we wrote helper methods to serialize our data to json and to organize our data into a primitive database in the filesystem.

2.3. UI

The user interface module had four important functions.

Display handling: Making sure that the display components of the interface are laid out in a spatially consistent and intuitive manner.
Control flow handling: Making sure that state and mode transitions are smooth and easy to follow.
User and timing event handling: Making sure that user commands (mouse clicks or speech commands) produce the expected results and that transitions occur at the correct timing intervals.
Kinect stream handling: Reading from the kinect to display skeleton and depth data as well as to sanitize and pass the skeleton data to the backend for classification.

To make sure that all four major functionalities of the UI are in agreement, the following simple finite state machine was used for control flow. See the **system description** section on more on the two modes.



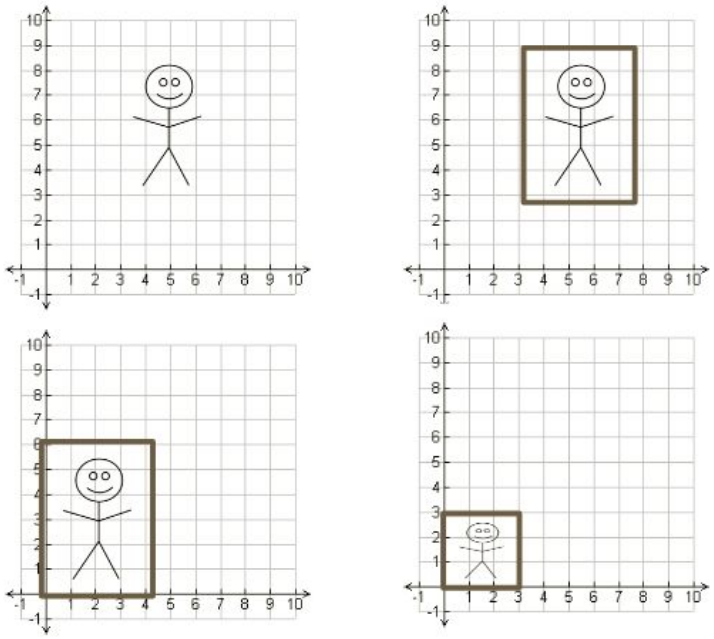
2.4. Back end (data storage and classification)

The back-end has two major responsibilities: data storage/retrieval and gesture classification.

2.4.1. Data storage

Our data format was based on the `Gesture` class from MiniProject2. We also added a `DataSet` class that maps vocab words to their translations and to their appropriate gestures. We set up a primitive filesystem-based database that our application can load from. This consists of an index of available datasets as well as the appropriate data files stored in json form using our own serialization functions. We also created an extensive set of helper functions to perform database operations, like making or saving a dataset, adding a new vocab word to a DataSet or adding a new gesture example. These helpers provided an interface to the front-end for loading data to show to the user and for saving the data back to the back end after a gesture has been performed. Our system saves data in a DataSet (stored temporarily in memory) every time the user performs a gesture in Practice mode. Data is saved back to the filesystem upon exiting the application.

Our data format also contains the ability to normalize gesture sequences. We found that gesture normalization was important to make our classifiers location invariant relative to the Kinect and integral for making Nearest Neighbors work well. We downsample all sequences down to 30 frames off the bat to limit the amount of data we have to transfer and store (as we realized later, having to save a lot of data was a significant UI slowdown). Our `Sequence` object also provides a `normalize` function which normalizes the gesture based on position. First, it calculates a "bounding cube" around the gesture by finding the high and low endpoints of the x, y, and z coordinates. It subtracts the corresponding min value from each coordinate of each frame so that one corner of the bounding cube is located at (0, 0). Then, the cube is scaled so that all side lengths are one. The picture below demonstrates visually how this is done, using a 2-d representation for simplicity.



Step 1: bound the action. Step 2: translate it to the origin. Step 3: normalize side lengths of the bounding cube.

2.4.2. Classification

We define a base `Classifier` class that exposes endpoints for setup, training, and classification, among others. We experimented with several classification methods which implementing this interface. Having a clean, generic interface let us easily swap out different classifiers when experimenting with our system to figure out which one worked best.

We ended up implementing several classifiers, but our Nearest Neighbor implementation (`NNClassifier`) worked best given the limited data we had available. This classifier requires no training, which was convenient because this meant that there was no lagging owing to training time whenever a new gesture was entered. When `classify(sample)` is called, the classifier compares the normalized sample to all of the gesture examples in the currently loaded database, keeping a heap of the most similar gestures. It returns the gesture that had a majority of the top n closest gestures, where n=3 since we expect around 3 examples per gesture.

Other available classifiers include the `NetClassifier` (implementing a simple neural net), `DTClassifier` (using a decision tree), and `Kmeans`.

2.6 Adding new vocab sets

Right now, we do not provide a UI endpoint to specify a new vocab set, as we decided to focus on designing for the actual learning process. However, the back-end process for creating a new

vocab set is trivial and consists of specifying a list of (vocab word, translation) tuples. An explanation for how to do this is included in the README of our project code. Once this process is complete, the user is ready to start training the system.

2.5. Speech Recognition

The speech recognition module is powered by the Python Speech Recognition library. This speech recognition library supports multiple backend speech engines including the CMU Speech recognition engine(Sphinx), the Google Cloud Speech API, the Microsoft Bing Voice Recognition, and the IBM Speech to Text, therefore some research had to be done to select a backend engine that works well and easily accessible.

Sphinx is the only engine in the list that works offline, so it was the first option to try since we don't want to always rely on fast internet. Sphinx ended up having very high error rate and picking up a lot of noise.

Most of the web and cloud based engines require access tokens and have daily call limits. The only major exception was the Google Web Speech API, which worked well enough with some parameter tuning and therefore became our backend of choice.

We also had to do some parameter tweaking to get the speech recognition to work well. We found that the Kinect microphone was more reliable than the one in our laptop. We also played around with two methods of recognizing speech: either having the library listen to a continuous stream and pick out words, or breaking the stream into discrete chunks and running detection on each interval. Listening continuously did not work very reliably, so we used the discrete approach with an interval size of 1 second, which was a reasonable tradeoff between reliably detecting commands and minimizing the latency of getting a response back to the user.

3. Performance

The main challenge of our system is handling pretty arbitrary gestures with little data (about 3 examples per word/gesture pair). We used our study to quantitatively evaluate how good our system is at this task. We replicated the use case as closely as possible: we gave each participant a vocab list of 10 words to learn, gave them the freedom to choose their own gestures, and had them practice each word three times before training. We then counted how many times our system accurately classified the gesture performed by the user during testing mode (note that this is different from counting how many times the system predicted a correct gesture, because our users sometimes got the word wrong). Louder than Words correctly classified 110/156 = **70.5% of gestures**.

3.1. Interesting failure cases

One interesting failure of our system was that it sometimes struggled to classify back-and-forth motions, such as repeatedly pointing at something or moving a hand back and forth across the stomach to demonstrate “scratching your stomach” (see pictures below). We think this is because that if such a motion is offset in time by even a short period, the gesture looks basically the same to a human, but it looks significantly different to a nearest-neighbor classifier (for instance, in the example below, the arm may be extended in the training sample while it is retracted in the testing sample and vice versa).

One of our test subjects performing a back-and-forth pointing gesture to demonstrate the word “you”.

Another interesting (but not surprising) failure was related to the fact that the Kinect does not track finger positions. We found through our user study that a surprising amount of gestures depend on finger position for their semantic meaning--for instance, a finger pointing towards versus away from the chest is the main difference between “you” and “me”, and a hand held with all fingers extended over the eyes versus the index finger pointed at the head can make the difference between “look” and “know”.

For this application of gesture recognition, we really want to give users the widest range of gestural cues possible to distinguish their different vocabulary words. We learned from our study that people seem to rely on both full-body and hand-specific gestures to convey meaning. Yet, neither of the devices used in this class (the Kinect or the Leap) provide the capacity to detect both. For our application, it would be really interesting to see if finger position could be used to disambiguate gestures that are skeletally similar.

4. Design Process

Originally, we envisioned that Louder than Words would be more of a classroom tool that could be built into language lessons. A teacher would pre-program the system with a set of vocabulary words and later on students could use it to quiz themselves on vocab. Although some aspects of ours system have remained the same (training on little data, user versus developer training of the system), other aspects have changed. This section describes some of the challenges, successes, and changes we made to the project during the semester.

4.1. Challenges

There were two major challenges we struggled with while working on this project. The first and the most significant roadblock was interfacing with the Kinect. After initial research, we intended to use either Pykinect or OpenKinect to transfer gesture data to Python. However, Pykinect did not work with either of our computers and OpenKinect only worked with the RGB and audio streams. The Kinect SDK accessed via Visual Studio C++ was the only interface that was working by the time we were presenting the prototype presentation. We ended up having the C++ code continuously stream skeleton data into a file and have python file read from it.

Fortunately, we were able to get a more reliable form of data transfer working by borrowing a Windows 10 Machine from IS&T which we were able to install Pykinect on. Since Pykinect itself is just the Python wrapper for a C++ library, the easiest switch was to abandon our prototype UI and build something around the Pykinect demo game.

The second major problem was that our UI was really slow and our backend used a lot of memory. This made our skeleton data detection and display really laggy, with sometimes a second of delay between a user's gesture and its registration in our UI. This was an application-critical issue because this meant that our system was actually recording and saving data from the wrong time window and was not accurately capturing the gesture.

We spent significant time debugging this issue and slimming down our code. We used code profilers to isolate pieces of code that were taking significant time. On the left of the green arrow, Figure 1 shows a recursive function (`Button.set_font_size`) in the UI that was taking a total of approximately 23.3244 seconds after it was called 107 times and on the right it shows the same function taking a total of approximately 1.072 seconds after 159 calls as result of improvements.

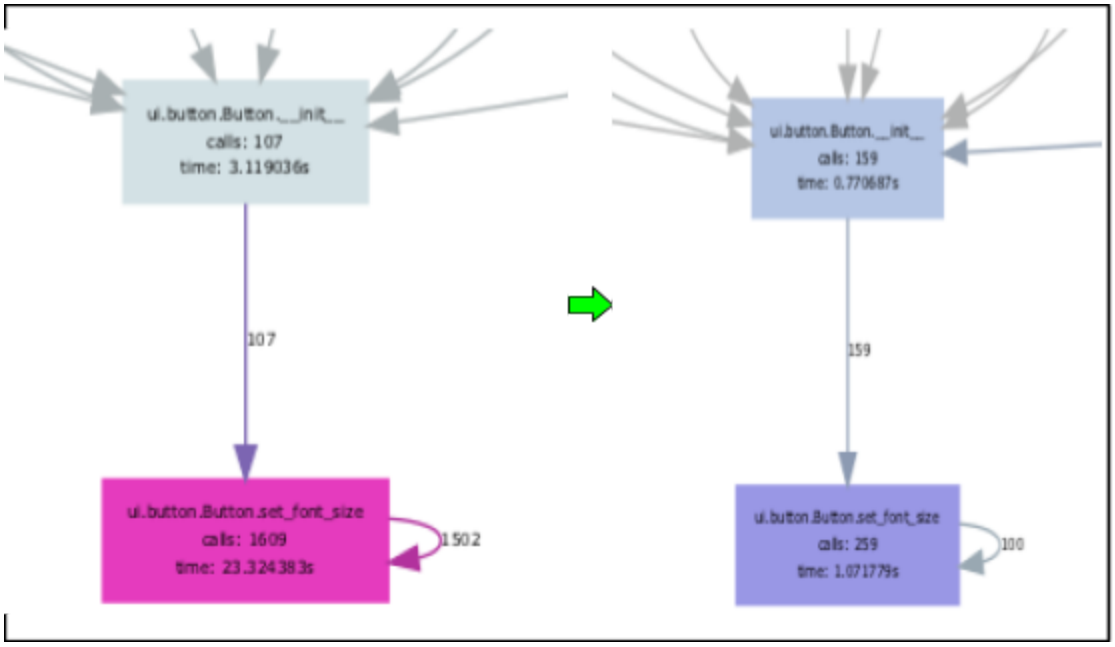


Figure 1

We also reduced the number of times the UI code was called without compromising any of the existing UI functionality or implementing supplemental code. For example, Figure 2 shows that on the right side of Figure 1 the UI was running for over 83 seconds while it was running for 38 seconds on the right.

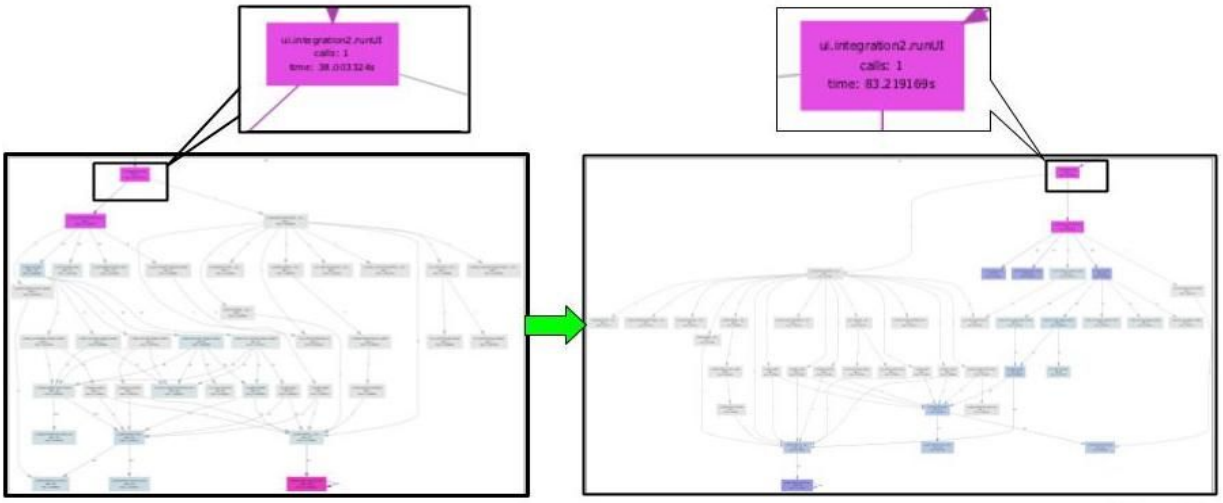


Figure 2

There were two major lessons to take away from this and we hope future students will use them to improve the performance of their programs. 1) It really matters where you put pieces of code in terms of performance 2) Please use code profilers especially when you're working with significant projects such as the 6.835 final project.

Finally, we realized that the way we were saving data was a significant source of slowdown, with up to 2 seconds being spent on serializing and saving gesture data as json. As a result, we decided to keep our dataset in memory and only save it back to the file system upon exit (using the Quit button, the quit command, or the exit button on the window), as opposed to saving to a file every time the system recorded a gesture. This sacrificed a little bit of data security (it's possible to lose some data if the system shuts down unexpectedly) for increased performance.

The profiler we ended up using was PyCallGraph.

4.2. Successes

One unexpected success was that the simple Nearest Neighbor classifier was the most effective classifier we tried (it was also the first one we tried, although we introduced modifications to our original setup). We also tried a Decision Tree classifier and a small neural network, but these did not have enough data to train effectively. The normalization applied to the NN classifier successfully made it position invariant.

Another big success was getting the control flow of our UI to be intuitive and easy to use. During the prototype studio, a lot of the critiques we received revolved around people being confused by the timing of the loop, not knowing when to perform the gesture, etc. We made some UI changes, shortened our gesture window from 4 to 2 seconds, and tested this aspect in our UI study to make sure we had cleared up the issue. We were happy to get mostly positive feedback. On average, people "agreed/strongly agreed" that the control flow was clear and the timing felt natural. They disagreed that they had too much or too little time to perform the gesture itself.

4.3 Pivots

4.3.1. Teacher versus student training

Originally, we envisioned that a teacher would train the system and that a classroom of students could all use it; that is, our system would have to work for an arbitrary user. However, during user studies we realized that the way people performed gestures varied a lot person to person, even if the gesture had already been demonstrated for them, which made classification really difficult. Thus, we decided to have the end user (the student) both train and test the system. This ended up working out well because during Practice/training mode, students had a chance to get familiar with the vocabulary and try to remember it before being quizzed.

4.3.2. Using English translations

We originally wanted to avoid showing the user a translation of the vocab term they were learning so that they ceased to associate the vocab word and a translation and instead associated the word with a gesture. However, we realized that with the student training the

system, we needed some way of indicating to the user what gesture they should perform given that they have not already memorized the foreign-language terms. Thus, we provide both the original term and its translation during Practice mode.

In training mode, if a user gets a word wrong, we provide feedback in terms of the term and translation of the correct word and the word that the system detected. This means that if the system makes an error (which we hope to avoid, but which we realize is inevitable no matter how good our classification), the user can explicitly identify the correct word. This lets the user continue learning during practice mode and avoids them inadvertently being told incorrect information by the system.

5. User Study

Here is a brief summary of our user study practice:

1. Give the subject an overview of our system and instructions for the study in the form of a script.
2. Present subjects with the list of vocabulary words they are going to learn. We give suggestions for a gesture for each word, but make it clear that users can choose their own gestures.
3. Have users practice the words/gestures in training mode for 3 iterations.In this mode, we video record the user performing the actions.
4. Have users test themselves for 3 iterations. In this mode, we video recorded the UI display to have a record of our system's inputs, outputs, and accuracy.
5. Give users a hard copy of the survey to fill out and open up the floor for more questions and comments about the system.

5.2. Results

We designed our study to focus on two main components of our system: the intuitiveness of the control flow and the accuracy of our classifier. As mentioned above, the accuracy was around 70%. The answers to our Likert questions indicated that our UI loop was not confusing for users.

Subject:	**System is helpful for learning new vocab**	**I was able to memorize words I was given**	**System was fun/ enjoyable**	**Preferred system to vocab list**	**control flow was clear**	**timing felt natural**	**length of time was too long**	**length of time was too short**	**System did well predicting gestures**
Rueben	5	4	5	5	5	5	2	2	4
Luis	4	4	4	4	4	4	3	4	4
Nathan	4	4	5	3	4	5	2	2	4.5
Prafull	5	4	5	5	4	3	2	3	2

Vivan	3	4	4	3	4	4	4	5	4
Barry	3	3	4	2	5	5	2	2	2
Average:	**4**	**3.833333333**	**4.5**	**3.66666667**	**4.33333**	**4.3333**	**2.5**	**3**	**3.416667**

Answers by subject to our Likert-style questions.

5.3. Other Takeaways

We learned that people generally found our system fun to use and did help people learn new vocabulary. However, multiple users said they still would prefer a normal vocabulary list because it is quicker to absorb than interacting with our interface and waiting for feedback. Thus, we think that our system could be helpful for younger kids (or older kids too!) who may be too impatient to memorize a lot of words and would be drawn to the game-like aspect of our system.

People expressed frustration that they were not shown the meaning of a word after they got it wrong in testing mode, especially if that was a result of a system misclassification.

Finally, since we're using the across-the-room version of the Kinect, people ended up standing pretty far away from the laptop while playing with the system, which made it hard to use the mouse to interact.

5.4. Changes made

In response to observations made during the user study, we made three main changes:

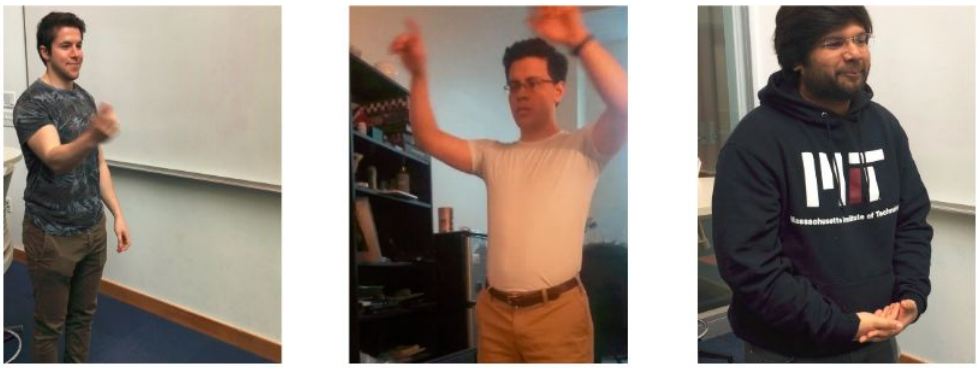
1) We made some further tweaks to our UI to make it clear when the system was recording based on suggestions we got from study participants, like changing the text of the word "GO!"from red to green.
2) When people got a word wrong, we started showing them the translation of the correct and predicted word. This lets users learn from their errors and, in the case that the system made an error, the user can still double-check the correct answer so they are not confused or mislead.
3) We added speech recognition for control flow so that users could interact with the system across the room.

6. What we learned

The most complicated solution is not always best. Our simplest classifier worked best for our use case.

If you run into a problem, look for a way to visualize what's happening: A day or two before we had to conduct the user study, we have realized that our skeleton was very laggy and sometimes taking full seconds. For a while we were bouncing around panicking and walking back changes or even introducing more changes that didn't help, until we have realized that we can visualize what's slowing down our system using code profilers(as discussed in the **challenges** section).

Gestures are very individual. We realized from our user study that the most "intuitive" gesture for a specific word varied a lot between people depending on the individual or their culture. For instance, we saw three different ways of performing the word "all" (picture below). We also realized that people can perform the same gesture in very different ways. These individual differences will be a challenge for any gesture recognition system.



Dhamaan (all)

Having an intuitive UI is important. When people could not figure out how to use our UI correctly because our timing or our visuals were non-intuitive, they would perform the gesture at the wrong time and our classifications would suffer as a result. Making it easy for people to use the system correctly was important for it to function properly.

7. The Next level

Our product was a proof of concept for a system that can be developed and deployed for a large scale use. To take it to the next level, the UI needs to be more functional and appealing, and the gesture classification more robust and data driven.

7.1. Next Level UI

A next level Louder than Words user interface should incorporate more modalities, add more states and modes to give the user more options and achieve the necessary functionality, and become aesthetically more appealing.

We think the system could be improved by adding more intuitive and rich ways to interact with it. For instance, adding the capacity to hear and pronounce foreign vocab words would be very beneficial. It would also be challenging because the system would have to be able to handle many different languages, meaning we would have to find a library capable of synthesizing and recognizing speech in arbitrary languages.

Another modality that might be helpful would be clicker control. An ideal clicker for Louder than Words would have four buttons for up-down and side-to-side movements for repeating or advancing to the next word as well as an “OK” or a “Select” button. This would be especially helpful since the user is often standing up straight in an open area away from the Kinect in possibly noisy environments and therefore speech recognition could fail while keyboard and mouse are hard to reach. Since the user would mostly be commanding the system to either move to the next state or go back to the previous, clicker control would introduce very little cognitive load.

7.2. Next Level gesture classification

Our current best classifier is nearest neighbor with size and position normalization. The reason is that we can’t require the user to input enough data for training adaptive algorithms such as the neural network family. This leaves us with very little room for improvements in two major ways: 1) besides struggling with offsets as discussed above(interesting failure cases), this classifier can do well on only sets of gestures that are quite distinct from each other. 2) the only significant thing we can do to improve that classification on those distinct gestures is to play around with the distance metric.

Thus, the next level Louder than Words needs to incorporate more adaptive and data driven learning algorithms including neural networks. One way to incorporate neural networks is to combine all the datasets from all the users to train the neural network, a decision tree, or another advanced algorithm. This would have to be done in such a way that common gestures for the same word can be recognized by the system, while outlier gestures from other participants do not interfere with recognition.

8. Responsibilities

Anelise had main responsibility over the back-end (gesture classification and data storage) and Mubarik had main responsibility over the UI including speech speech recognition. We worked closely together on many aspects of the project, such as brainstorming, debugging, deciding on features, running the user study, and writing this report.

9. Conclusion

Louder than Words provides an engaging and memorable way to learn vocab words in a new language. Although more work needs to be done to perfect the system, we have already seen

through our user study that we are achieving our main goal of helping people learn vocabulary while having fun.