ASL-to-Text

Group Members: Kevin Dibble, Jasmine Moran, Susan Souza

Repo: [https://github.com/uml-ubicomp-2016-spring/ASL-to-Text](https://github.com/uml-ubicomp-2016-spring%20/ASL-to-Text)

The goal of this project was to use the LeapMotion device to develop an extremely basic American Sign Language translator to text, beginning with the alphabet for a proof-of-concept, and to determine the feasibility of adding simple and then more sophisticated signs further on if using the LeapMotion. Future applications for this type of application would be for a portable interpreter for sign languages that would offer live sign-to-speech capability to the user. The vast majority of americans do not understand even the most basic of ASL and because of this, the speech impaired must rely almost exclusively on an interpreter or slow written communication. A portable computerized translator would allow the speech impaired to interact with the general populace directly in a comfortable and efficient way.

Core features for this short iteration were to detect a specific sign from a user's hand and have the corresponding letter print to the screen. We were also able to add a method to break on pauses, which is intended to delimit words more clearly than the app’s initial behavior of just printing a giant string of characters matched to the screen. Further potential features were discussed (but not implemented)and included things like piping the matched strings of characters to other programs for use(text editors, chat programs), having a tutorial, having a voice component to ‘read’ the text matched, having actual simple words rather than just letters and having further tutorials to use as a teaching tool. Also very briefly considered and discarded for the purposes of this short project were dynamic punctuation and predictive spell-checking/word completion.

Project design [visual diagram of project components/flow] :



File structure: ASL-to-Text

├── lib

├── data

| ├── gestureData.csv

| ├── gestureDataSimple.csv

| └── gestureDataLong.csv

├── src

| ├── asl2txt.py

| ├── leapASL.py

| ├── matchASL.py

| └── gui.py

└── utils

├── machineLearningTest.py

├── shiftingTests.py

└── printVectors.py

/lib: resources like the Leap library (Leap.py)

/data: our training data and superfluous testing data

/src: actual app file

/utils: debug, experiments with techniques and tools, data collection utility

asl2txt.py: [Susan]

* This file starts up the gui controller and acts as a gateway into everything else. While this file simply provides an entrance point into the project, the added verbosity allows for a more clear code structure. To run, type on the console:

python asl2txt.py

gestureData.csv: [All]

* This file holds some of the initial datasets we intended to use as training data for the machine learning algorithm.

gestureDataSimple.csv: [Kevin]

* This file holds the data from gestureData.csv, trimmed to only hold one frame of data for our initial full test of the machine learning algorithm with collected data.

gestureDataLong.csv: [All]

* The third iteration of the training data for the machine learning algorithm, this file is parsed by matchASL.py and fed into the machine. Each data entry consists of 29 frame’s worth of data that matches to a specific letter. For gestures that can be detected in multiple positions, multiple entries are added.

gui.py: [Susan]

* gui.py is an MVC implementation for a GUI for text display using the Tkinter package. Controller (Ctrl) instantiates an aslListener, gets the data from the Model to the View, and provides a route from the Listener to the Model. The model for MVC in this case holds data from the listener, which reports changes to the controller; the view is the displayed text that is updated when the controller is signalled to a change in the Model.

leapASL.py: [Kevin, Susan]

* This file contains the aslListener class, an extension of Leap Listener, which interfaces with the LeapMotion. The aslListener takes frame data from the LeapMotion, extracts finger vector data, and builds a running list of 29 frames (a little less than a second) of data. This data is then passed to the machine learning algorithm and the corresponding result is sent to the Controller (which passes it to the Model) to be displayed for the user.

matchASL.py: [Kevin]

* This file contains the scikit-learn machine learning algorithm interface. A helper procedure (getGestureDataFromFile) is included to parse the training data file and create a NumPy array to train the algorithm with. The algorithm itself has an interface to simplify the interactions with the scikit-learn system and succinctly determine gesture matches from user input.

machineLearningTest.py: [Kevin]

* This debug file was used during my research into the machine learning algorithm. Various tutorials and sandboxing were run in here so as not to clobber any of the project’s working code.

printVectors.py: [Kevin, Susan]

* This data-gathering file was created to simplify the data gathering workflow. Starting as a simple printout of each frame, this helper grew into its current state where it will append new entries into the data file itself.

shiftingTests.py: [Susan]

* This debug file was used to quickly double-check the method to be used for shifting the sets of 29 frames-worth of data to make room for a new frame’s-worth of data to pass to the scikit-learn machine in leapASL.py.