Jim Version 2.4.1

With Implemented Speech Recognition and Voice Synthesis Capabilities

**Materials (Hardware)**:

Jim (An Affective Platform for A.I.)



*Figure 1: A picture of the exterior of the robot Jim who was used as the basis for this project. He is angry because of all the attention he lost due to the snowstorm.*

Xtion PRO LIVE (Similar to Microsoft’s Kinect)



*Figure 2: A picture of an ASUS Xtion PRO LIVE. It is equipped with a microphone along with an RGB depth sensor. This device will ultimately be Jim’s eyes and ears.*

**Introduction**:

Jim is currently an affective machine constructed out of Legos and fleece with custom made electronics completely designed by students. Recently there was software written in Java that is able to setup a USB to MIDI connection between the computer and the custom made circuit board such that MIDI messages can be constructed, sequenced and ultimately sent to the PIC microcontroller. The microcontroller’s software, which is written in C, interprets the received MIDI messages in such a manner that allows for the operation of Jim’s five Lego motors and two Lego lights. The current version of Jim can only construct a sequence of MIDI messages with manually determined timings and send them to the PIC while synchronously playing a pre-recorded audio file. While this worked well for demonstrational purposes, I wanted to improve his abilities by making his capabilities more flexible, especially after he was extremely well received at the Artificial Intelligence Conference in Quebec City this past summer. Specifically I desired to give Jim the ability to be able to interpret spoken language, construct a response based on a given auditory input using a glorified Eliza program and finally respond to the interacting individual in a similar auditory fashion through text-to-synthesized speech.

**Project Goals**:

Ultimately there were three main sub goals to this project. The first was to program an Xtion such that we could use its microphone to accept audio input from an auditorily interacting individual and send that to a speech-to-text converter. Unfortunately, the language that is used to code the Xtion is Processing and while this language is very similar to Java, it cannot be compiled or executed using Java’s compiler or application launcher. Processing does however come with .jar files that would allow one to use Processing libraries in a Java program. Therefore another task we needed to accomplish was to successfully link Processing’s libraries in addition to the third party Processing speech-to-text (stt) libraries to the java compiler and java application launcher.

Our second sub goal was to construct an improved and specialized Eliza program that would accept the returned string from our speech-to-text converter as input. Specifically, we wanted to make an Eliza program that is more controlling over the conversation in order to more accurately predict what the human would say making it easier for us to construct reasonable and intelligible phrases for Jim to use as responses. Secondly, we wanted to specialize our Eliza program in emotional conversations because of Jim’s eventual implementation as an affective platform for artificial intelligence.

Finally, our third main sub goal was to implement a text-to-speech converter to convert the returned string received from our specialized Eliza program into “spoken” audio. An added challenge to this part of the project was the need to acquire the timings at which the words would be spoken such that the necessary MIDI messages could be sent to the PIC microcontroller to move the mouth synchronously with the spoken audio.

**Results**:

I downloaded and installed all the necessary drivers, software and libraries to be able to program the Xtion which include OpenNI, the NITE wrapper, the PrimeSense Kinect package1, Processing and the stt libraries2. While OpenNI automatically added its org.OpenNI.jar file to the CLASSPATH environment variable, I had to manually add Processing’s core.jar file and the stt.jar file to my computer’s CLASSPATH which successfully allowed me to use Processing and stt libraries in Java applications.

In the end I wrote three java files as shown in the **Appendix** section. The first being Ears.java which extends PApplet such that it allows one to run a Processing application in a Java application as if it were being run in a Processing IDE. The second file was DisplayFrame.java which displays the PApplet application in a JFrame so that the application can execute. Finally testEars.java was written to get everything going by instantiating a new DisplayFrame. Together, these chunks of code allow the computer to convert spoken audio into text. Specifically, the computer’s default microphone accepts audio input when any key on the keyboard is held down. Once the key is released, it is sent to Google’s Web Speech API and the interpreted text along with the confidence level of the translation is then displayed to the screen.

A few tests were run and it was determined that the translations are usually quite accurate with a confidence level generally above 0.9 with a perfect confidence level of exactly 1.0. A few sample translations are displayed in the **Appendix** section under “test.txt”. The three statements that were given to the program to translate were: “Hey Jim, how are you doing?”, “Which witch it which?” and “Their shoes are not there, but they are over there by their shoes.”. There are a few things to note about the translations. In addition to getting all the words correct, the program was also able to disambiguate which word was appropriate to use at the appropriate place in the statement as seen in the second and third translations. While this is extremely helpful for our purposes, one downside is that all the translations automatically lack structural punctuation. For example one has to say “question mark” in order for the program to place a question mark at the end of the sentence. Unfortunately, because of this, some important information from what is originally uttered is lost. However the program’s ability to disambiguate between words and its ability to translate statements with a generally high level of confidence makes this a good program to help us complete the second and third sub goals.

**Conclusion:**

It was determined that Google’s Web Speech API along with Processing’s stt libraries will be adequate for testing and constructing Jim’s speech recognition and voice synthesis programs due to its ability to disambiguate words with a high level of confidence. However, further testing is needed in order to determine how other factors affect the translation process such as accents, ambient noise, and ones proximity to the microphone.

For future purposes, we would need something that does not require access to the internet and that can provide for a translation more quickly than what Google’s Web Speech API can provide. One possibility is that we could use something along the lines of Nuance’s Dragon NaturallySpeaking software which is said to provide very fast translations while also being quite accurate.

**References**:

-The website at which the software was ultimately acquired from.

1<https://code.google.com/p/simple-openni/wiki/Installation_PreOpenNI2>

-Was the website where the speech-to-text libraries were downloaded. Unfortunately, recently further development has been stopped.

2<http://stt.getflourish.com/>

**Appendix**:

***\*\*\* Ears.java \*\*\****

/\*

A simple voice recognition program. Even though this current version does not use the Xtion's microphone, implementing it shouldn't change the code that much.

\*NOTE\*: Due to the changes of the CLASSPATH environment variable, this program was entirely written in a Processing IDE which could now be simply copied and pasted over without any issues.

\*/

**import** processing**.**core**.**PApplet**;**

**import** com**.**getflourish**.**stt**.\*;**

public class Ears **extends** PApplet **{**

STT stt**;**

String result**;**

public int width **=** 800**;**

public int height **=** 150**;**

public void setup**()** **{**

size**(**width**,** height**);**

//Google Web Speech API key.

String key **=** "AIzaSyAHq1A\_uRvRU-fEppb0IVI9qCxvRXw\_KdI"**;**

// Init STT with default manual record mode.

stt **=** **new** STT**(this,** key**);**

stt**.**enableDebug**();**

stt**.**setLanguage**(**"en"**);**

// Some text to display the result.

textFont**(**createFont**(**"Arial"**,** 24**));**

result **=** "Say something!"**;**

**}**

public void draw**()** **{**

background**(**0**);**

text**(**result**,** 50**,** 50**);**

**}**

//Method is called if transcription was successful

public void transcribe**(**String utterance**,** float confidence**)** **{**

println**(**utterance**);**

result **=** utterance**;**

**}**

public void keyPressed**()** **{**

stt**.**begin**();**

**}**

public void keyReleased**()** **{**

stt**.**end**();**

**}**

**}**

***\*\*\* DisplayFrame.java \*\*\****

/\*

The class that displays the Processing application in a JFrame similar to how Processing automatically displays the launched application in a window when run in its native IDE. So far as I know, we need to have a JFrame running in order to run the Processing application.

\*/

public class DisplayFrame **extends** javax**.**swing**.**JFrame **{**

public DisplayFrame**(){**

//Could also work for this purpose although I am not entirely sure of the difference

//between the line below and the line with the Ears instantiation.

//processing.core.PApplet sketch = new CircleSketch();

Ears ears **=** **new** Ears**();**

**this.**setSize**(**ears**.**width**,** ears**.**height**);** //The window Dimensions

setDefaultCloseOperation**(**javax**.**swing**.**WindowConstants**.**EXIT\_ON\_CLOSE**);**

javax**.**swing**.**JPanel panel **=** **new** javax**.**swing**.**JPanel**();**

panel**.**setBounds**(**20**,** 20**,** 600**,** 600**);**

panel**.**add**(**ears**);**

**this.**add**(**panel**);**

ears**.**init**();** //this is the function used to start the execution of the ear

**this.**setVisible**(true);**

**}**

**}**

***\*\*\* testEars.java \*\*\****

/\*Simply the class that gets everything going.\*/

public class testEars **{**

public static void main**(**String**[]** args**)** **{**

**new** DisplayFrame**().**setVisible**(true);**

**}**

**}**

***\*\*\* test.txt \*\*\****

03:07:04 Transcribing

03:07:07 Recognized: hey Jim how are you doing (confidence: 0.9825246)

hey Jim how are you doing

03:07:07 Listening

03:07:13 Transcribing

03:07:15 Recognized: which witch is which (confidence: 0.95343125)

which witch is which

03:07:15 Listening

03:08:16 Transcribing

03:08:20 Recognized: their shoes are not there but they are over there by their shoes (confidence: 0.9757292)

their shoes are not there but they are over there by their shoes