A Report
on

Speech Recognition Using Arduino and Raspberry Pi

By
Johnny Esquivel

Prepared in partial fulfillment of
the course CSCI 552
ADV Microcontroller Electronics
Taught by: Dr. Charles Rogers

at

Texas A&M-Commerce

April 27, 2014

TABLE OF CONTENTS

Introduction

Microcontrollers are small computer systems on a chip designed for a specific real-time task. Arduino is a microcontroller, originally designed for the education community, is an easy to use, single board microcontroller. It runs C/C++ like programs called Arduino Programming Language and comes with a simple integrated development environment (IDE). The key features that the Arduino possesses that are needed for this project is a USB serial port, input output pins (I/Os), and the analog to digital converter (ADC). The ADC can measure an analog voltage from a sensor and convert the continuous value into a discreet digital value. In our case, a microphone converts audio or sound pressures into voltage, but the voltage that is continuous between zero volts and the supply voltage. We can then use the Arduino to convert this voltage into an integer value to send off for processing.

Raspberry Pi is a credit card size computer running Linux. It has all the same features as a general computer system like the ability to connect to a monitor, keyboard, and mouse. It also has a USB serial port, network capabilities and a SD card slot for storage. The Linux operating system that we use on the Raspberry Pi is Raspbian [1], a combination between Debian Linux and Raspberry Pi. Raspbian comes with the capability of running python programs.

An artificial neural network is a computer model that is based on the brain's neural network (NN). The concept start with an input stimulus being feed into a network of neurons that are connected to other neurons. The network ends at an output, and the value at the output is the desired result. The connection between each neuron has a different weight or bias associated with it. The way that NN is conditioned to give the

desired result at the output is to train the NN. This network can be trained by giving the network a series of inputs with a desired result. The difference or error between the desired result and the result obtained by the NN is then back-propagated to the weights and biases of the neuron connections to reduce the error. This type of NN, which is used in this project, is called a back-propagated NN.

Speech recognition is the process of a computer system capturing and interpreting spoken words. The goal of this project is to capture voice from a microphone using the Arduino, then interpreting the voice using the Raspberry Pi. The vocabulary of the speech recognition is limited to the words “red”, “green”, and “blue”. When the Arduino send an audio sample to the Raspberry Pi, the Raspberry Pi will decide which word was most likely spoken and light up a corresponding LED.

Method

The method used to interpret the speech was derived form a study by John-Paul Hosom, Ron Cole, and Mark Fanty [2], but was greatly simplified since the vocabulary of this project was only three words. As shown in ***Figure 1***, the process begins with the

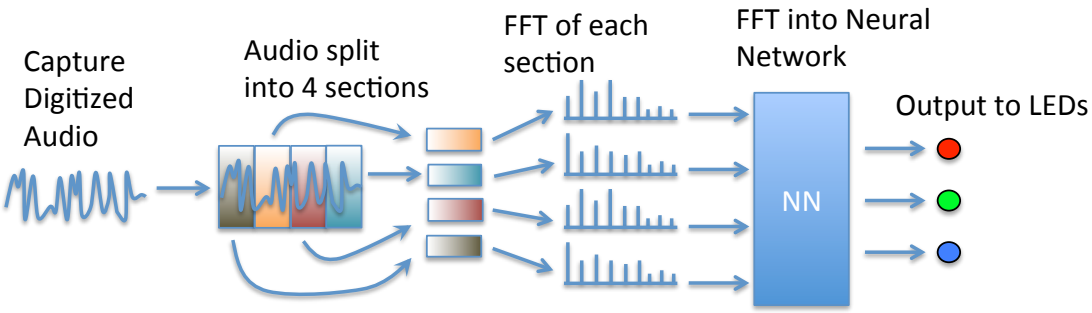

Figure 1. Speech Recognition Process.

captured digitized audio, which is performed by the ADC in the Arduino. The audio sample is split into 4 content windows. The idea of the context windows is to split a word into its separate phonemes. A phoneme is "The smallest contrastive linguistic unit which may bring about a change of meaning" [3]. More complex techniques such as that described by Hosom et. al., involve computing the size of the context window to capture complete phonemes. In this project the three words (red, green, and blue) are distinct enough to use a fixed context window size, which is ¼ of the audio sample size. The frequency content of each context window is calculated using a fast Fourier transform (FFT) algorithm. The frequency content is essentially what sounds make up the content window. The frequency content is then passed into the neural network (NN). The NN is trained to understand the three words and light the LED that matches the word. Conceptually the NN can find the pattern in the frequency content data that is different between the three words.

Hardware Design

Converting sound into voltage is accomplished by an Adafruit MAX9814 electret microphone module. The module has a MAX9814 auto gain amplifier that will reduce the amplitude of the output voltage waveform if the sound is too loud or increase the amplitude if the sound level is weak. As shown in ***Figure 2***, the module's output is connected to the analog input (A0) of the Arduino. The module is supplied by the 3.3V output of the Arduino. Originally the quality of the sampled audio signal was very poor

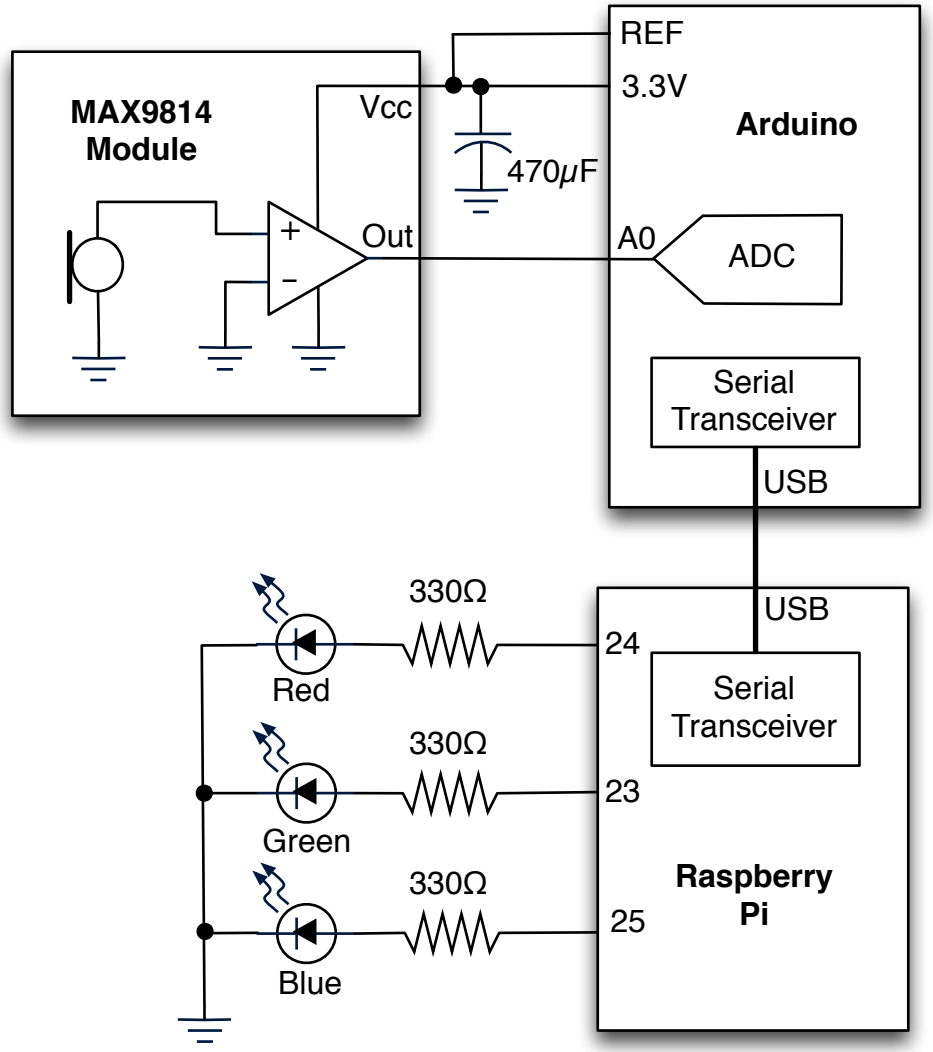

Figure 2. Voice Recognition Schematic

due to the noise in the system. Two circuit changes were added to reduce the noise. The first change was to add a 470µF capacitor to the 3.3V supply line to bypass the noise to ground. The second change was to connect the reference pin of the Arduino to the 3.3V passed the capacitor. That way any noise from the 3.3V supply that is not removed by the capacitor, that makes it to the output of the microphone module, will be ignored because it is seen on both the output and the reference. The reference line and the supply line were twisted to further aid the cancelling out of any noise as shown in ***Figure 3***

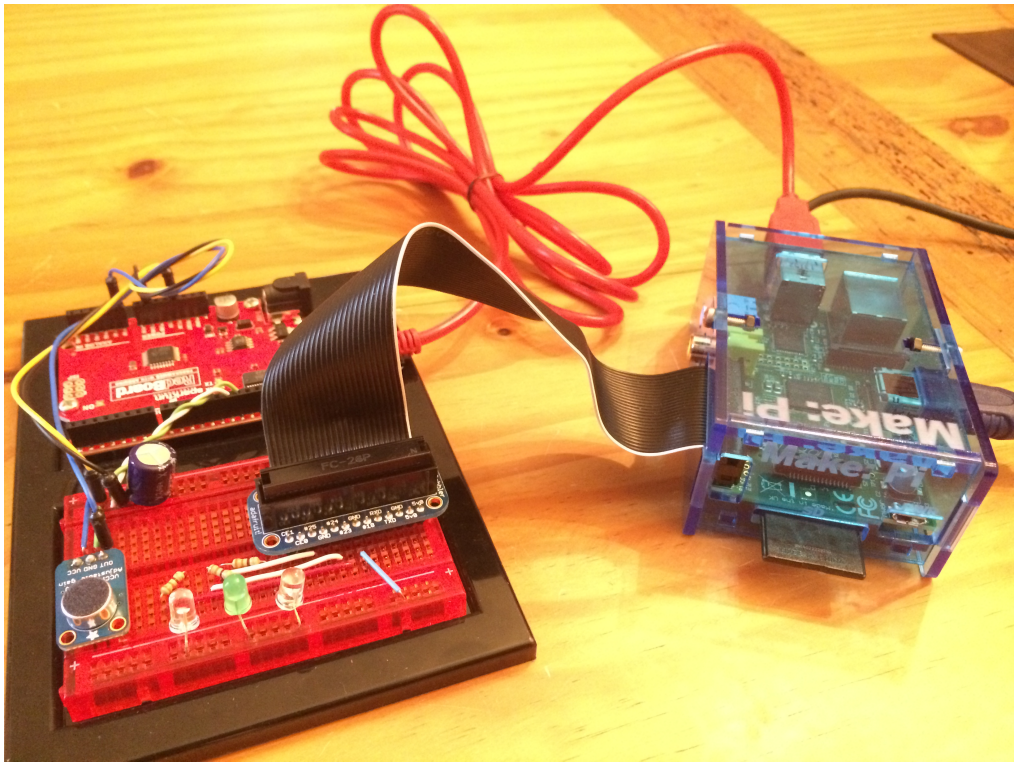

Figure 3. Voice Recognition Hardware

The Arduino does the analog to digital (A2D) conversion of the microphone voltage. The digital voltage values are sent to the Raspberry Pi via the USB serial connection. FFT conversion and neural network processing is done in the Raspberry Pi. Once the Raspberry Pi decides which word was spoken, it lights up the appropriate LED. The LEDs are connected to the Raspberry Pi through a 330Ω resistor in order to limit how

much current the Pi must put out to light the LED. The 330Ω resistors are connected to the general purpose I/Os (GPIOs) of the Pi. When the Pi wished to turn on an LED, 3.3V is applied to the GPIO, and it turns off the LED by applying 0.0V.

Arduino Programming

The goal of the Arduino program is to convert the analog voltage from the microphone module to digital values and send them to the Raspberry Pi for processing. The entire Arduino program is listed in appendix A. The key function that converts the analog voltage into digital is the "analogRead(pinNumber)" function. This function samples the pin given as pinNumber and returns an int value between 0 and 1024. By changing a handful of setting, shown in the program, the A2D conversion can occur approximately every 100µs or at a sampling frequency of 9kHz. The human voice range is from 50Hz to 3.5kHz however most of the energy lies between 300Hz and 3kHz[4]. According to the Nyquist sampling theory [5], 2 times 3.4kHz or 6.8kHz sampling frequency should be sufficient to sample the microphone voltage. However, even though the Arduino can sample at 9kHz, the Arduino can't send data to the Raspberry Pi at 9kHz. Also, the Arduino only has 1KB of RAM so it can't store the samples and send them later. To start with, the default serial baud rate is 9600 baud or symbols per second. Therefore the first modification to the program was to increase the baud rate to between the Arduino and the Raspberry Pi to 11,520, as shown in ***Table 1***. The baud rate was

Table 1. Serial Communication Speed.

Baud	Data Type	Fs (Hz)	BW (Hz)
11520	Character	2374	1187
23040	Character	3156	1578
23040	Binary 8bit	6804	3402

further increased to 22,040. Since the result of the analog read was 0 to 1024, this number was a 10-bit number, meaning in binary it is represented by 10 bits. By truncating this number to 8-bit, or in other words, keeping the 8 most significant bits, the analog read is reduced to a single Byte. Generally the serial communication protocol is

sending characters, which are 1Byte each. For instance, if we send the number 1024 across the serial port, we are sending 5 bytes, one byte for each character (1, 0, 2, 4, \n). Now instead of sending a character to represent each character in the analog read number, we could send the entire 8-bit binary number in one character packet. One difficulty that arises when sending pure binary number is where one number ends and the next begins. To solve this problem we send a 'H' character or 00001000b every 4 bytes to synchronize the Arduino and the Raspberry Pi. Although the 'H' character is overhead, the overall sampling speed increases to 6804Hz and therefore the audio bandwidth we can receive increases to 3402Hz, which is all of the human voice range.

The Arduino sends a 2048 sample packet to the Raspberry Pi. It starts sending the packet when an audio threshold is reached. In the line of code:

```
if ((sample[bufEnd] < 452) || (sample[bufEnd] > 572))
```

the Arduino is waiting till the analog voltage goes above 572 or below 452, where 512 is the no sound level. Once the threshold is crossed, the Arduino begins sending the packet. The purpose of this threshold is to give the Raspberry Pi an audio sample that has the start of the speech at a consistent location, that way the audio can latter be split into four pieces at roughly the same position each time. The threshold level that was chosen does a good job at rejecting random noise, however, when the threshold is surpassed, the spoken word is already well underway, so some of the beginning of the word can be missed. To solve this problem, the Arduino was programmed with a circular buffer that is continuously being refilled with the latest microphone readings. Therefore when the sample is sent, it begins with the buffer, which was the last 200 samples before the threshold was crossed.

Raspberry Pi Programming

The goal of the Raspberry Pi program is to interpret the audio sample from the Arduino as red, green, or blue and then light the appropriate LED. To accomplish this goal, the Raspberry Pi is split into four separate program files, all of which are listed in appendix B.

The first program is the main.py program that is the entry point to all of the programs. It is basically a menu to choose what function you wish to run. The options are:

- Exit the program
- Record samples for a dataset
- Build a back-propagation model
- Use current prediction model

"Exit the program" does as you would think, it terminates the program. "Record samples for a dataset", takes the user though speaking and recording audio samples, which will later be used to build a NN. The program file responsible for this function is buildDataSet.py. After the dataset is generated the "Build a back-propagation mode" function is used to build the back-propagation neural network (BPNN). The file, buildModel.py, is the program used to build the model. Finally, the program useModel.py is what is called if the use selects "Use current prediction model", which will continuously monitor the serial communication with Arduino. If a sample is presented, it will determine what word was said and light the appropriate LEDs.

Data Set Creation

The creation of a dataset begins by asking the user, how many sample they wish to take. Basically this means, how many time the user will speak one of the three words into the microphone. Next the program asked the user what label to add to the dataset. For instance if the user answer three and red, then the user will speak the word red, three time into the microphone. The program will create three text files called red_0.txt, red_1.txt, and red_2.txt with FFT data from each recording. The location of the files is set by the workingDirectory variable at the top of the main.py file that the user can change as necessary. The FFT is performed using the FFT function from the scipy python library and therefore must be previously installed. Building the dataset is completed when the user has generated samples for each of the three words.

Build Back-Propagated Neural Network

Once the dataset is completed, the use can choose the build a BPNN. PyBrain python library must be installed to build the BPNN. The user is asked how many sample to use in building the BPNN. Of course there must be enough samples from each color to build the model. Next the user is asked for the max number of iterations, which is the maximum number of iterations to train on, though convergence could be reached before this number. The training algorithm uses 75% of the samples to train on and the other 25% to test the model on. Once the training is completed, a neural network model is build and written to the file fnn.xml in the directory specified by the workingDirectory variable at the top of the main program.

Using the Current Model

Once the BPNN model has been build, the use current prediction model function

can be started. It will read the fnn.xml model and construct a network. Next it begins listening to the serial port for any new audio samples. If an audio sample is sent from the Arduino, then the Raspberry Pi converts it to four FFTs and sends it to the BPNN. The result from the BPNN is three number which total 1.0. The three numbers correspond to the three colors and the number is the percent likelihood that what was said is that color. The Pi then chooses the most likely color and turns on the LED of that color. To apply voltage to that LED, the Pi uses the RPi.GPIO python library. At this point, the program continues to loop, listening for another sample from the serial port until the end of time.

Results

There are numerous ways of scoring how well a predictive model works. The two methods that are used here are the error and the confusion matrix. The error is simple the number of results that were predicted incorrectly over the total number of results. After training is completed the program will report the training error and test error. The training error is how many results are predicted incorrectly from the training data and the test error is how many of the test results were predicted incorrectly. The confusion matrix gives more granularity. It shows the results from each class or group and what it was predicted to be.

The initial dataset created to test the program was 10 samples for each color. A BPNN model was generated using 20 iterations or epochs to build the model. The resulting model stats were as follows:

epochs: 20 train error: 8.70% test error: 57.14% build time: 2m 37s

Table 2. Confusion Matrix for 10 Samples.

Confusion Matrix				
		Predicted		
		Red	Green	Blud
Actual	Red	7	2	1
	Green	2	7	1
	Blue	0	0	10

A test of the entire system was conducted by using the current predictive model and speaking into the microphone. Out of 20 attempts, the system only predicted correctly 8 times. The result was low but this test was only used to look for bugs in the system. Since no bugs were found a larger sample test could now be conducted.

The next sample set was 300 samples for each color. A BPNN model was generated using 200 epochs to build the model. The resulting model stats were as follows:

epochs: 200 train error: 0.89% test error: 2.22% build time: 4h 17m 08s

Table 3. Confusion Matrix for 300 Samples.

Confusion Matrix				
		Predicted		
		Red	Green	Blud
Actual	Red	299	1	0
	Green	0	296	4
	Blue	1	5	294

A test of the entire system was conducted by using the current predictive model and speaking into the microphone. Out of the 30 attempts, every attempt was predicted correctly, however time between samples must be given for the Raspberry Pi to refresh the 200 sample circular buffer. Also note that in order to build this more accurate model, a larger number of samples and iterations was required, which took 4 hours to build.

Conclusion

In this project, the Arduino and Raspberry Pi were successfully employed as a voice recognition system, though with a limited vocabulary. The key challenges identified during the design were the limited bandwidth for transferring data between the Arduino and the Raspberry Pi and the limited memory of the Arduino. To overcome this challenge, the Arduino to Raspberry Pi communication was changed to binary communication instead of characters, the digital value was reduced from 10 to 8 bits, and the speed was set to the highest communication speed. Finally, a test was run in which a neural network was built from the 300 samples. The error rate of the model was 2.2% meaning it only predicts incorrectly 2.2% of the time. More samples and training time could be used to reduce the error even further if desired.

References

[1] Raspbian. Retrieve Apr 5, 2014 from BYTE Mark Hosting Website: http://www.raspbian.org

[2] Hosom, J.P., Cole R., and Fanty, M., (1999) *Speech Recognition Using Neural Networks at the Center for Spoken Language Understanding*. Retrieve Apr 2, 2014 from Oregon Health and Science website: http://speech.bme.ogi.edu/tutordemos/nnet_recog/recog.html

[3] Gimson, A.C., Cruttenden, A., (2008) The Pronunciation of English (7ed.), Hodder

[4] Munir, B., (2012) Voice Fundamentals – Human Speech Frequencies. Retrieved from the Unified Over IP website: http://www.uoverip.com/voice-fundamentals-human-speech-frequency

[5] Nyquist Frequency Reterieved Apr 26, 2014 from Wikipedia website: http://en.wikipedia.org/wiki/Nyquist_frequency