95

Small-Scale Speech Recognition on a Microcontroller

Sanjana Subramanian, ss6169 *Mechanical Engineering, Fu Foundation School of Engineering and Applied Science*
Columbia University
New York City, USA
ss6169@columbia.edu

***Abstract*—The implementation and advancing of speech recognition functionality in various consumer technologies has been made possible via the development of signal processing, machine learning, and low-cost embedded processors. The purpose of this project was to implement a rudimentary speech recognition system onto a low-cost microcontroller. This would allow for greater accessibility in everyday machines, greater access to communication and education for children and adults who cannot read or write, and easier communication across language barriers. A simple "yes" "no" speech recognition system was set up with an Adafruit Circuit Playground Bluefruit and a corresponding TFT Gizmo, which was able to process and respond to audio samples in a reasonably short amount of time, if it was not as accurate as hoped for.**

I. Introduction

Speech recognition, also known as automatic speech recognition (ASR), is a technology that allows computers to transcribe spoken language into written text. This technology has a wide range of applications, from virtual assistants to transcription services, and has the potential to revolutionize how humans interact with machines and each other.

In the past few decades, speech recognition has proved to be much more than a novelty. One study, by Fu et. al., shows that drivers who text with voice-to-text technology or a virtual assistant are less in danger of a head-on collision than drivers who text manually [1]. Natural and efficient communication between humans and machines is not only convenient, but life-saving.

Speech recognition is also important via its potential to improve accessibility for individuals with disabilities. For example, individuals who have difficulty typing or using a keyboard due to physical limitations may find speech recognition to be a more convenient and accessible way to communicate with computers. In addition, speech recognition can also be used to transcribe spoken language into text for individuals who are deaf or hard of hearing, allowing them to access written content more easily. Accessibility is also increased for able-bodied people. Portable and easily accessed transcription services allow for easier translation between people who speak different languages, and children too young to read and write effectively on their own are able to self-educate and discover via speech-to-text search systems.

While generally we see speech recognition utilized on purely software-based platforms, he new prevalence of low-cost processors, largely on microcontrollers, and an increase in accessible educational resources on speech recognition and embedded systems means we can hope to leverage speech recognition implementation in new and exciting ways.

Using microcontrollers in system design has various key benefits. On major benefit is how small and inexpensive microcontrollers components are. Since all components are integrated onto one chip, boards are relatively cheap and easy to manufacture. Microcontrollers are also incredibly modular, and their RAMs, ROMs, and I/O ports can easily be adapted to any specific project. However, microcontrollers do come with drawbacks. The amount of memory available for program data and supplementary material is limited, and microcontrollers cannot directly interface with high-power devices. Nonetheless, it is useful to examine how we may be able to use microcontrollers to add speech recognition as an element in more complex integrated systems, as in mechatronics and robotics–this integration has the potential to make machines both more accessible and more interactive. This paper seeks to explore rudimentary implementation of speech recognition in inexpensive microcontrollers.

II. Speech Recognition Framework

Most speech recognition systems follow a similar workflow:

1) Pre-processing: The audio signal is first pre-processed to remove noise and enhance the signal-to-noise ratio. This can involve filtering, normalization, and other techniques.
2) Feature extraction: Next, the system extracts a set of features from the audio signal that are relevant for recognizing the spoken words. These features might include the spectral content of the signal, the pitch and duration of the spoken sounds, and other acoustic characteristics.
3) Modeling: The system then uses these features to build a statistical model of the spoken language. This model is typically a probabilistic model, such as a Hidden Markov Model (HMM), which represents the probability of different sequences of words being spoken given the observed features.
4) Decoding: Once the model is trained, it can be used to decode the spoken words by finding the most likely sequence of words that corresponds to the observed features. This is typically done using algorithms such as the Viterbi algorithm or the forward-backward algorithm.

5) Post-processing: Finally, the system may apply various post-processing techniques to improve the accuracy and fluency of the transcribed text. This can include techniques such as language modeling, which uses the statistical properties of the language to predict the most likely words or phrases given the context of the transcribed text.

A. Phonemes, HMMs, & GMMs

Speech recognition can be performed at signal, phoneme, and word levels. Statistics, Fast Fourier Transforms, and complex mathematical models are used to extract features of audio data and return probabilities of intended words and phrases. One of these models is the Hidden Markov Model, or HMM. In the context of speech recognition, an HMM represents the process of generating speech sounds. When humans speak, they produce a stream of sounds that can be broken down into a sequence of distinct units called phonemes. For example, the word ”cat” is composed of three phonemes: /kæt/.

The hidden states in the HMM correspond to the phonemes in the word, and the observations correspond to the speech sounds produced when the word is spoken. By training an HMM on a large dataset of speech sounds, it is possible to learn the probability distribution over phonemes given the observations, which can then be used to recognize spoken words.

A Gaussian mixture model (GMM) is a statistical model that represents a distribution over a set of observations as a mixture of multiple Gaussian distributions. In the context of speech recognition, GMMs can be used to model the distribution of speech sounds, such as phonemes, produced by different speakers.

To use GMMs for speech recognition, we first need to train the model on a large dataset of speech sounds from multiple speakers. This involves estimating the parameters of the Gaussian distributions that make up the mixture, such as the mean and covariance, based on the observations in the training data. Once the GMM is trained, we can then use it to classify new speech sounds by computing the probability of the observations given the model. By comparing the probabilities computed by the GMM with a set of known phonemes, we can determine which phoneme is most likely to have been spoken based on the observed speech sounds. Using GMMs and HMMs in tandem improves the accuracy of a speech recognition system.

B. Mel Frequency Cepstral Coefficients (MFCCs)

After obtaining an audio sample, we need to extract certain features to identify key components of the audio and reduce background noise. Mel frequency cepstral coefficients (MFCCs) are commonly used as such features. To compute them, a Fourier transform is performed on the audio to convert it from a time domain to a frequency domain. The resultant spectrogram is then transposed to the Mel scale, which mimics the human auditory system–sounds that show up as equidistant on the Mel scale would seem the same distance away to the human ear. As MFCCs are low-dimensional, we can use them to fairly easily separate background noise from the speech signal [2].

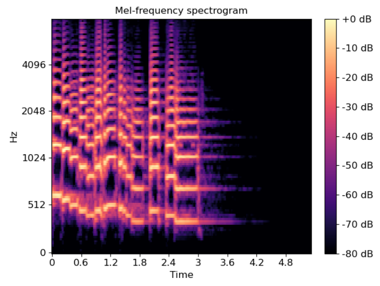

Fig. 1. Mel Scale Spectrogram

C. Linear Predictive Coding

Linear predictive coding (LPC) is a method for representing a digital signal as a set of linear equations that can be used to predict the signal’s future values. The goal of LPC is to model the signal in a way that allows it to be reconstructed with a minimal amount of error.

LPC is useful because it can be used to compress the size of digital audio or speech signals, which can be useful for storage or transmission. It works by representing the signal as a set of coefficients that describe the relationship between the signal’s current and past values, rather than storing the signal’s actual values. This allows the signal to be reconstructed with a minimal amount of data, resulting in a smaller overall size.

D. Arduino IDE Libraries

So far, we have covered how speech recognition can be done ”from scratch”, or purely mathematically. However, as computer science and machine learning becomes more accessible, this work does not need to be done from scratch. The primary library used in this project was TensorFlow, an open-source library for machine learning and artificial intelligence. TensorFlow Lite is a lightweight version of TensorFlow designed for mobile and embedded devices. TensorFlow Lite was developed to enable the deployment of machine learning models on-device, meaning no cloud connection is necessary. It is designed to be fast and lightweight, with a small footprint and low latency, making it well-suited for real-time applications.

III. Mechanical System Overview

In this project, speech recognition was added to a decorative wall-mounted sculpture. The sculpture was manufactured with a FormLabs resin 3D printer, and consisted of a humanoid face, a digital model of which is shown in Figure 2. The end goal was to control the sculpture backlight using ”yes” and ”no” keywords, although the speech recognition function was tested using a TFT microcontroller screen and set of microcontroller-mounted neopixels. Figure 3 shows the printed sculpture with a white backlight.

An electronics components holder was integrated into the face to secure all circuit elements. The table below shows the

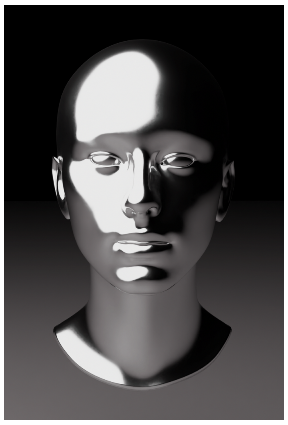

Fig. 2. Mechanical Design Rendering

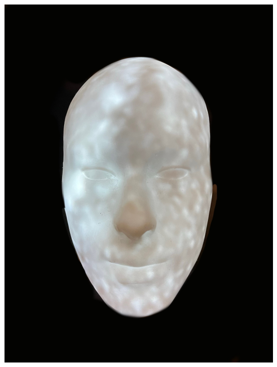

Fig. 3. Mechanical Design

electronic components and quantities to be incorporated into the robot circuit.

Item	**Quantity**
Adafruit Circuit Playground Bluefruit	1
TFT Gizmo	1
Battery Holder 4xAA	1

The Adafruit Circuit Playground Bluefruit (CPB) is a microcontroller development board based on the popular Adafruit Circuit Playground platform, and is equipped with a variety of sensors and input/output (I/O) devices, such as a microphone, a speaker, and 10 programmable neopixels. The CPB is programmed using the Arduino programming language, but also supports CircuitPython [3].

The microphone in the Circuit Playground Bluefruit acts as a biometric, and captures the speech signal for processing. The amplifier inside the microphone amplifies the electric signal to increase the Signal-to-Noise ratio, and then sends the amplified signal to the analog-to-digital converter in the CPB for sampling and segmentation. Each segment, or frame, is then processed to obtain various parameters for comparison. According to the literature, it can be expected with components of these specifications (or comparable specifications) that the system will process frames in around $60ms$ and, therefore, will reach a verification decision in as little as 4 times the duration of the speech signal. The Circuit Playground Bluefruit is shown in Figure 4.

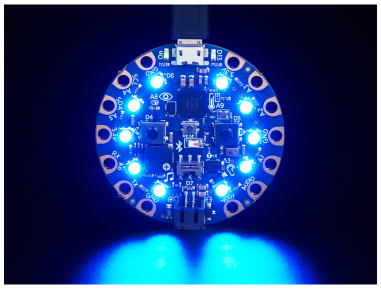

Fig. 4. Circuit Playground Bluefruit

The TFT Gizmo, as shown in Figure 5, is a small, portable display device that uses Thin Film Transistor (TFT) technology to display images and text. The Gizmo is specifically designed to be compatible with Circuit Playgroung microcontrollers. TFT displays are known for their high quality, wide viewing angles, and fast refresh rates, making them well-suited for use in a variety of applications. In this project, the TFT Gizmo was used as an intermediate visual interface to test program function [4].

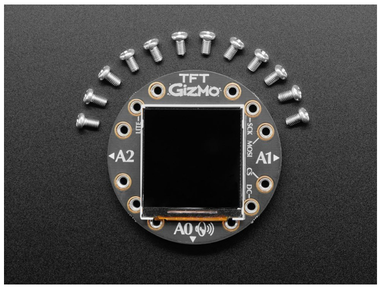

Fig. 5. TFT Gizmo

During intermediate testing, the expected outcome was for the TFT Gizmo to display a bitmap of either the word ”yes” or the word ”no”, and for the CPB neopixels to light up green or red (respectively) in response to the word being said. Three question marks would appear on the Gizmo if the audio was too noisy or unrecognizable.

IV. Methods

For programming on the Adafruit Circuit Playground Bluefruit, some setup was required. Reference [5] provides some guidance on how to do that, specifically for TF Lite implementations. Aside from downloading the latest Arduino IDE and the Adafruit nRF52 support packages, various libraries needed to be installed for either ML implementation or graphical outputs. Libraries are listed as follows:

1) Adafruit Arcada
2) Adafruit NeoPixel
3) Adafruit FreeTouch
4) Adafruit Touchscreen
5) Adafruit SPIFlash
6) Adafruit Zero DMA

7) Adafruit GFX
8) Adafruit LIS3DH
9) Adafruit Sensor
10) Adafruit ImageReader
11) ArduinoJson
12) Adafruit ZeroTimer
13) Adafruit TinyUSB
14) Adafruit WavePlayer
15) SdFat (Adafruit Fork)
16) Adafruit Fork

While it is necessary for all these libraries to be installed, the Arcada, NeoPixel, SPIFlash, and GFX libraries were especially important in this project as they allowed for control over image output, bitmap reading, and access to digital pins on the board.

The TensorFlow Lite library also needed to be downloaded. For this project, the 1.15.0 version was used (not the pre-compiled option), as it is very well documented. It can be installed with the following line in the command prompter:

```
pip install tensorflow=1.15.0
```

Of course, this needs to be installed in the libraries directory of the Arduino folder. For Macs, the path usually looks like

```
Users\[USER]\Documents\Arduino\libraries
```

.

A pre-trained model from TensorFlow was used for this project. It can be found at the following link:

```
https://github.com/tensorf
low/examples/tree/master/lite/examples
/sound_classification
```

Bitmaps and detection files were taken from Reference [5]. After compressing the TensorFlow model using TensorFlow Lite, an Arduino program was written to respond to various results with visual outputs. Code from Reference [6] was referenced for troubleshooting and guidance.

V. Evaluation

The program was able to run fairly successfully on the Circuit Playground Bluefruit. It was tested using the CPB/Gizmo mechanical design in a computer lab, where background noise was especially prominent. In a test run of thirty trials, the program was able to recognize key words as according to the information in the table below. ”Yes” and ”No” were each key words for ten out of thirty trials. The remaining ten trials were random simple monosyllabic words (can, pet, lid, etc) that were *not* key words.

Key Word	**% Accuracy**
Yes	80%
No	40%
Not a Keyword	90%

In the noisy setting, it was much more likely for the program to accurately respond to ”yes” than ”no”. It is possible that the experiment was biased towards the word ”yes” because it is simply easier to say and recognize (even as a human, much less as a program). The word ”no”, when said while congested or with certain conversational inflections, can very easily sound like /dō/. This is one intuitive explanation for the program’s miscalculation. It is possible that using an online recording of the key words, rather than a live person, would have rendered the program more accurate. It is worth noting that it would only be more accurate *for that online recording*, though, which is not a viable use-case for any speech recognition implementation.

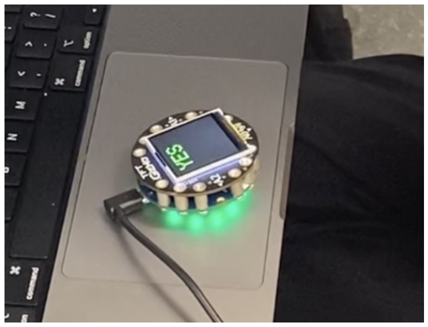

Fig. 6. ”Yes” Recognition, Gizmo

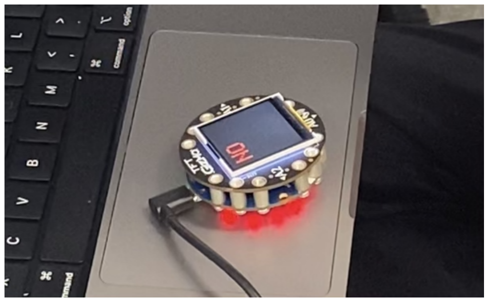

Fig. 7. ”No” Recognition, Gizmo

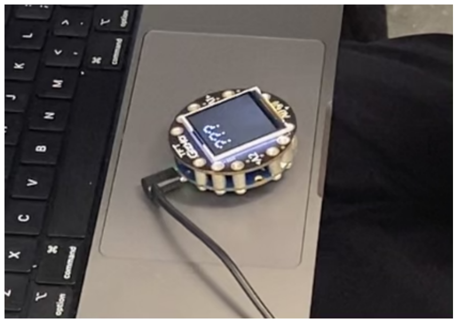

Fig. 8. Indeterminate Recognition, Gizmo

Testing was redone in a quiet setting, with almost no background noise. This time, instead of using the Gizmo as a graphical interface, the resin sculpture/neopixel mechanical design was used. The neopixels would provide a red backlight in recognition of "no", a green backlight in recognition of "yes", and white background for indeterminate results, as seen in Figure 3. Once again, thirty trials were performed in the same manner as they were for the noisy setting. The results for these trials are displayed in the table below.

Key Word	**% Accuracy**
Yes	90%
No	70%
Not a Keyword	100%

In the quiet setting, the program was far more accurate than in the noisy setting. Intuitively, a reduction in background noise would result in a significant difference between "feature" signals and "non-feature" signals. In the noisy setting, important signal features were more likely to get confused with background noise, especially on a program with a low-power, rather than high accuracy, prioritization in its first iteration.

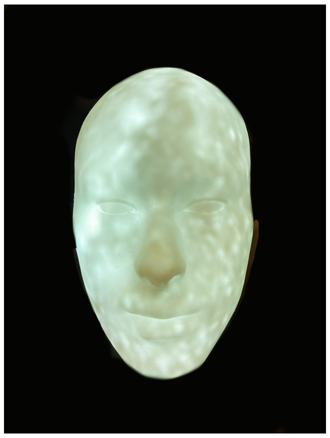
Fig. 9. "Yes" Recognition, Neopixels

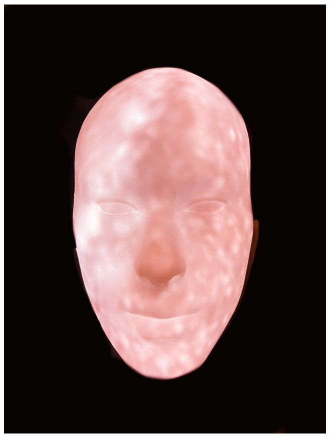
Fig. 10. "No" Recognition, Neopixels

To statistically guarantee the noisy trials and the quiet trials were truly two separate sets of data indicating varying performance, a Two-Sample Independent t-Test was performed via equations 1 and 2. If the magnitude of t_{exp} was greater than the magnitude of $t_{\alpha/2,\nu}$, the null hypothesis that the performances were statistically the same could not be rejected.

$$t_{exp} = \frac{\bar{x}_1 - \bar{x}_2}{\sqrt{\frac{S_1^2}{N_1} + \frac{S_2^2}{N_2}}} \tag{1}$$

$$\nu = \frac{(\frac{S_1^2}{N_1} + \frac{S_2^2}{N_2})^2}{\frac{(\frac{S_1^2}{N_1})^2}{N_1 - 1} + \frac{(\frac{S_2^2}{N_2})^2}{N_2 - 1}} \tag{2}$$

Because the magnitude of t_{exp} was less than the magnitude of $t_{\alpha/2,\nu}$, the program was statistically proven to have varying accuracies in different environments.

Another key thing to note about the performance was the response time. Because the key words were all monosyllabic, it took around .2 seconds for a human speaker to say them. Based on noisy trial video footage timestamps, it took around 1 second after the end of a key word being spoken for the program to response with a "yes", "no", or indeterminate bitmap and correlating neopixel color. A processing time of around 5 times the duration of the speech signal is fairly quick, especially for circumstances involving only monosyllabic words or commands.

VI. Limitations & Further Innovation

A significant roadblock to the effectiveness of this project was background noise. A noise suppression filter to enhance speech recognition may have proven helpful. Especially because microcontrollers have limited power and resources to effectively manage and run speech recognition programs, recognition errors are more likely to occur when the environment is noisy.

Speech recognition on microcontrollers can be used in a wide variety of applications, including in assistive technology, in human-machine interaction, and in educational settings. Speech recognition can also be used to build assistive technology for people with disabilities, such as devices that allow people with mobility impairments to control devices using their voice; it can be used to build more natural and intuitive interfaces for interacting with machines, such as robots or industrial equipment. As machine learning and embedded systems technologies become more and more accessible, the accuracy of low-cost, low-power embedded speech recognition systems is likely to rise. This will continue to render ASR as not just a novelty, but as a tool to develop equity and education on a global scale.

VII. Reproducibility Information

For the UF2 to be successfully flashed onto the Circuit Playground Bluefruit, Circuitpython needs to be flashed onto the board first. A helpful tutorial for doing this is available at the following link:

https://learn.adafruit.com/welcome-to-circuitpython/installing-circuitpython [7].

References

[1] X. Q. Fu R, Chen Y, "A comparative study of accident risk related to speech-based and handheld texting during a sudden braking event in urban road environments." https://www.ncbi.nlm.nih.gov/pmc/articles/PMC7459486/.

[2] Librosa, "Feature extraction." https://librosa.org/doc/main/.

[3] A. Industries, "Circuit playground bluefruit - bluetooth low energy."

[4] M. Leblanc-Williams, "Adafruit circuit playground tft gizmo." https://learn.adafruit.com/adafruit-tft-gizmo?view=all.

[5] L. Ada, "Tensorflow lite for circuit playground bluefruit quickstart." https://learn.adafruit.com/tensorflow-lite-for-circuit-playground-bluefruit-quickstart?view=all.

[6] G. L. Callie Yim, Diana Chou, "Voice recognition on simple microcontrollers." http://www.cs.umd.edu/ dchou/papers/818w$_{p}aper.pdf$.

[7] K. Rembor, "Welcome to circuitpython!." https://learn.adafruit.com/welcome-to-circuitpython/installing-circuitpython.