The Report Committee for Yifan Gong
Certifies that this is the approved version of the following Report:

Enhancing Touch Interactions with Passive Finger Acoustics

APPROVED BY
SUPERVISING COMMITTEE:

Edison Thomaz, Supervisor

Danna Gurari, Co-Supervisor

Enhancing Touch Interactions with Passive Finger Acoustics

by

Yifan Gong

Report

Presented to the Faculty of the Graduate School of

The University of Texas at Austin

in Partial Fulfillment

of the Requirements

for the Degree of

MASTER OF SCIENCE IN INFORMATION STUDIES

The University of Texas at Austin

May 2019

To my family, for their love and support.

Acknowledgments

First and foremost, I would like to thank Edison Thomaz, my report supervisor, for giving me a chance to join Human Signals Lab and do research with amazing people, for sparking my interests in academic research, and for encouraging me to think more and overcome obstacles. I really appreciate his concern.

Thank you to Danna Gurari for guiding me to the machine learning world, for serving as a co-supervisor to this report, and for teaching me how to be a researcher.

Thank you to Jakki Bailey and James Howison for teaching me to face difficulty bravely, and for helping me developing professional skills.

I would like to particularly thank Richard Cao for hours spent discussing and working on this project, and for making research time fun. This report would not come to life without his help.

Thank you to Rebecca Adaimi, Sarnab Bhattacharya, Keum San Chun, Mona Sachdev and Dawei Liang for sharing their knowledge, explaining concepts and giving me suggestions when I was struggling for problems.

Lastly, thank you to Zihan, Huaixi, Qihui, Alexandra, Lori and people in dorm 614 for being my friends, for encouraging and believing me all the time, and for bringing me thousands of happy times.

Enhancing Touch Interactions with Passive Finger Acoustics

Yifan Gong, M.S.I.S.
The University of Texas at Austin, 2019

Supervisors: Edison Thomaz
Danna Gurari

In this report, we introduce a unique method of interacting with mobile devices through passive finger acoustics. Phone users typically use one of two fingers when interacting with a touchscreen device, leaving the other fingers idle. The motivation is that users can make use of their idle fingers to enhance the experience of an application. By wearing minimally obtrusive rings on the thumb and index finger, users can make distinguishable clicking sounds to quickly perform actions, without having to interact directly with the screen. Our system leverages the microphone embedded in the mobile device to capture sound and recognize sound in real-time without requiring Internet connection. The different sounds introduce new ways for users to interact with their devices, without cluttering the screen. We evaluated our system on an Android drawing application, which allows the user to switch tools based on clicks. We optimized our system based on accuracy, classification speed, and ease of use, in order to create a comfortable user experience.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

Mobile devices are widely used in everyday life. The traditional interaction method between humans and mobile devices is commonly based on touching interfaces. Nowadays, diverse input techniques such as touch, sound, acceleration, etc. make the interaction between human and mobile devices flexible. For example, users can simply press on a point, or long press for a different function, and even zoom in and out with two fingers. However, due to the small size of the touchscreen, there are restrictions on the amount of direct physical interaction that can be done.

Previous works have explored unique ways of expanding the types of interactions that are possible with a mobile device. Several approaches [5] [24] [43] present novel solutions to distinguish different types of finger touch including the contact area of a thumb and tapping with different parts of a finger. However, these modes require extreme precision, and are difficult to perform consistently. More recent works take advantage of the dexterity of human fingers. This opens up another avenue of options for finger interaction. Typically, only one of two fingers are simultaneously used with a touch screen. When one finger is involved in a touch event, the remaining fingers are in an

idle state and free to move around. Building on this idea, Zhang and Lim have combined touch and gesture to create richer type of finger input methods [49] [31]. These works track the movement of fingers by attaching motion-tracking chips to fingertips. This technique allows fingers to perform actions without touching the screen. The downside of these methods is that wearable sensors on fingers must be powered and that data collected by sensors need to be transferred to the device. Consequently, both of these are major limitations and prevent wearable sensors from being used freely.

An approach to utilize idle fingers more freely is to create more space for fingers to interact with. Since the small size of the screen limits the number of ways that multiple fingers can interact through touch, recent works have explored expanding the space of finger interaction. Techniques such as SideSight [7], SkinInput[25], Skin Touch[28] and Lumiwatch [47] offer solutions for sensing user touch around the screen to extend the space of interaction surface. Their methods embedded certain sensors on devices to capture special signals. These have advantages over tracking methods since fingers are individually more free, due to less restriction from wearables. The disadvantage is that fingers will naturally have to touch the device to interact with it, making the fingers very cluttered around the device itself.

We present an approach, which takes the problem in another direction. Our system requires users to wear minimally obtrusive rings on the fingers which can generate passive acoustic information. Our system maintains the advantages presented by Zhang and Lim, by providing the fingers the free-

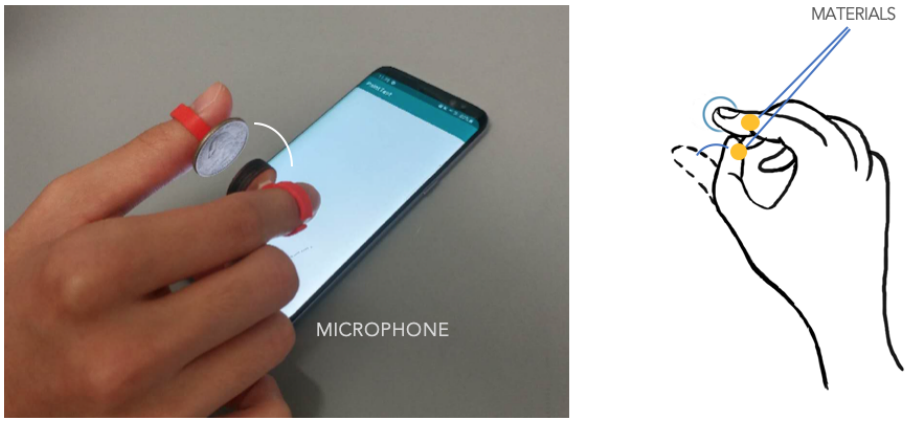

Figure 1.1: Our system leverage the microphone on the mobile device to capture sound produced by rings on fingers and recognize them.

dom to move anywhere near the device, while removing the disadvantages of requiring the wearable sensors to be powered. Our system leverages the existing microphone embedded on mobile devices to capture sound. By using the rings to create specific, distinguishable sounds, the mobile device will recognize these sounds and respond accordingly.

We evaluated our system on an Android drawing application. A typical drawing app on a small phone screen will often designate the entire screen to drawing, with a small button that can be tapped to open a panel of tools. This requires the user to tap a button to open the panel, tap the desired tool to choose it, and then tap somewhere to close the panel. Using our system, the user can assign specific tools to their corresponding sounds to quickly access those tools with a single click. For example, if a user is drawing with two colors,

blue and yellow, and needs to alternate between them quickly, our system is an efficient way to greatly reduce the number of total actions in the drawing process.

The main contributions of our work are:

- We design a system to enhance touch-based finger input by making use of idle fingers. Our system recognizes the sound created from tapping on different materials and makes actions.
- We minimize wearable devices on fingers to simple rings. The wearable devices in our system maximize freedom for fingers and comfortable to wear.
- We implement a real-time, offline sound recognition system and demonstrate that one-time click sound on different materials can be recognized by machine learning methods.
- We discuss the opportunities and challenges in real life.

Chapter 2

Related Works

Our work is mainly inspired from Touch+Finger[31]. The goal of our work is to use sounds to make secondary fingers interact with touch-based mobile devices when only the primary finger is actually in touch interaction. We reviewed approaches that track finger movements beyond the touchscreen. Some novel input technologies consider acoustic information to detect gestures. Other works present the technique of sound recognition, which our idea is closely related to.

2.1 Input with a Ring

Many innovative systems have been designed to detect finger gestures. These methods are implemented by tracking finger movement through the use of wearable sensors. Rings with build-in sensors are worn on the finger which is a good place to collect data. For instance, Lim et al. present a technique to detect multi-finger gestures to augment traditional touch interaction by wearing IMU sensors on idle fingers [31]. As the fingers move, the sensors will send data through USB connection to an external PC. iRing[37] used an infrared sensor to capture finger inputs. Ringeraction[20] detected

thumb-based gestures by wearing hardware on the index finger. Microphone as the most common sensor has been widely used in tracking finger movement. FingerOrbits[50] tracked thumb movement with an inertial measurement unit and a microphone. FingerSound detected thumb gestures by wearing a microphone and a gyroscope on the finger [49]. FingerPing[51] mounted a surface transducer for emitting sounds and four microphones as receivers to capture sound signals. By analyzing four unique frequency responses, their system can recognize different gestures. Researchers also utilized magnets to track the position of fingertips[48] [11] [10] [2]. By wearing sensors on the fingertips, they can compute its position relative to the magnet placed location.

2.2 Finger Interaction Beyond Screen

Other related approaches detected in-air finger gestures go beyond screen by capturing specific signals. These types of approaches do not require wearable devices on fingers. However, the downside is that they need to make modifications to the hardware of mobile devices such as mobile phones or smart watches to capture the respective signals. Another device that requires hardware modification is SideSwipe, which used a GSM signal to detect finger gestures around mobile devices [53]. The researchers designed a circuit board and added it to the mobile device to capture a unmodified GSM signal. Serendipity[45] integrated motion a sensor to smart watches to detect finger and hand gestures. WristFlex[16] used an array of force resistors to detect finger pinch gestures. A particularly unique approach was FingerIO, which

transformed a mobile device into a sonar system to capture inaudible sound signals [36]. It was able track the position of fingers by analyzing the signal. ViBand developed a custom kernel to improve the existing accelerometer embedded in a smart watch to capture bio-acoustic signals [29]. They used the bio-acoustic data to classify hand gestures. In addition, computer vision technology has been widely used to track fingers and recognize gestures[8] [26] [42], such as OmniTouch[23] and Air+Touch[12], etc. They utilized a camera attached on the shoulder or wrist to detect finger behavior. Electrical sensing has also been used in tracking fingers, such as SkinTrack[52].

2.3 Acoustic Input and Sensing

Some approaches leveraged sound signals to recognize gestures. SoundWave[22] utilized the embedded speaker and microphone of computing devices to sense user gestures, for example, hand waves, by detecting the shift in frequency of a sound wave. TILES[18] is designed to track audio activity using a mobile device for human subject studies. Amento et al. [1] built a wristband-mounted system to recognize gestures by using a small piezo-electric microphone to sense sounds produced by finger gestures. Mujibiya et al. [34] designed an armband with ultrasound transmitters to sense on-body touch.

2.4 Sound Recognition

Machine learning models have been increasingly used to recognize sound. Several researchers have explored using features to recognize sounds[13] [14].

Works also demonstrated the possibility of recognizing sounds in real time and producing a particular response. Some works [38] [40] developed applications to recognize environmental sounds in real-time leveraging microphones on mobile devices. Researchers have utilized sound recognition results on human activities studies. Ubicoustics [27] uses VGG16 convolutional network to classify sounds and recognize activity based on the sound classification results. Chen et al. [9] proposed a bathroom activity monitoring system based on sound. In addition to recognizing environmental sound, researchers demonstrated that impulsive sound can also be recognized using machine learning methods[17] [4].

Chapter 3

Methods

In this chapter, we describe concepts used in the system and provide a detailed description of them.

3.1 Audio Recording

The AudioRecord class allows Java applications to access audio input hardware to record audio. It differs from MediaRecorder class in that AudioRecord is able to record the raw data of an audio signal and keep it in the buffer, which can further be used for processing.

3.2 System Design

Mobile recognition systems can be implemented in different ways [38] [40]. One way is by extracting features and recognizing sound on mobile devices directly. Using this method, the system does not require a wireless connection with other devices. However, it does restrict the scale of the system. The other way is to use a back-end server that receives data as it is read. Feature extraction and sound recognition are both done on the cloud. This way requires Internet to transfer data. But a remote server can give more power

on computing. In addition, these two ways can be combined to some extent. Either the feature extraction stage or classification stage can be finished on a device or a server. In our case, because the availability and usability of the system is important, we chose to implement feature extraction and recognition on the mobile phone directly to avoid transmitting data over an Internet connection.

3.3 Feature Extraction

Feature extraction is used to convert an audio signal to a vector consisting of important audio properties. These feature vectors are used to train machine learning classifiers. Choosing suitable features based on audio signals is an important aspect to building a robust learning model. Acoustic features can be divided into two categories, temporal and spectral features, which refer to features found in the time and frequency domains, respectively, of an acoustic signals.[38] Different audio features can reflect different characteristics of sound including pitch, timbre, and frequency. For example, temporal features typically express the energy of a signal, whereas spectral features are applied to identify pitch and rhythm. The characteristics of click sounds are different from environmental sound, voice and speech. Click sound is short in time and its frequencies is in the higher part of the spectrum. Based on these properties, we choose to use the Mel-frequency cepstral coefficient (MFCC) and Zero-Crossing Rate as features to build our model.

3.3.1 Mel-frequency cepstral coefficient (MFCC)

MFCC is a commonly-used frequency domain feature in audio classification [39]. It optimizes the simple way of converting a time domain signal to frequency domain signal by mimicking the function of a human ear. MFCC uses Mel-scale filter bands after applying fast fourier transform (FFT) to the signal[32] where higher frequency filters are greater than lower frequency filters [15] since cochlea has more filters on low frequency signals. The following formula can be used to convert frequency in Hertz to Mel frequency scale:

$$Mel\,(f) = 2595log\left(1 + \frac{f}{700}\right)$$

MFCC is traditionally used for identifying linguistic content[33] [35], but in general, it is a powerful technique for sound recognition[44] [30] [6].

3.3.2 Zero-Crossing Rate (ZCR)

ZCR is a time domain feature that captures the number of times that a signal crosses the zero axis. Zero crossing rate (ZCR) is defined as such:

$$ZCR = \frac{1}{T-1}\sum_{t=1}^{T-1}\prod s_t s_{t-1} < 0$$

Within a fixed size buffer, a higher frequency signal is likely to exhibit a higher Zero crossing rate. Zero crossing rate has been heavily used to classify percussive sounds [21].

3.3.3 TarsosDSP

TarsosDSP[41] is an open-source Java library designed for audio processing. It features a wide selection of feature extracting methods, such as a percussion onset detector, pitch detector, MFCC, zero crossing rate, etc. TarsosDSP also provides simple and easy-to-use implementations of these methods using pure Java, which is advantageous in the fact that it doesn't require external dependencies. TarsosDSP is available for Android too, which makes it a great option for real-time audio processing.

3.4 Classification

Classification is used to determine the type of sound based on input features. In recent works, three types of classical machine learning models have performed remarkably on sound recognition [38]. They are the Multilayer Perceptron, Random Forest, and Support Vector Machine (SVM) classifiers. A Multilayer Perceptron is a feedforward neural network that excels at supervised learning. A Random Forest is an ensemble of Decision Trees trained via the Bootstrap Aggregation method[19]. Lastly, SVMs can efficiently perform non-linear classification using the kernel trick, which creates enormous flexibility in the model. All three of these are examined in the evaluation process of the system.

3.4.1 Weka

Weka[46] is a collection of machine learning algorithms, designed using Java. It excels at a large spectrum of tasks, including clustering, classification, and visualization. Furthermore, Weka has built-in models for the Multilayer Perceptron, Random Forest, and Support Vector Machine (called "SMO" in Weka) classifiers, which immediately can be trained with specified hyperparameters on an input dataset. Weka for Android is the Android implementation of Weka, with a few limitations on hyperparameter tuning, making it a powerful choice for machine learning tasks on Android as well.

Chapter 4

Implementation

In this chapter, we describe the system architecture, software and hardware components in the system and open source tools that enable feature extraction and classification.

4.1 Motivation

Our system requires users to wear minimally obtrusive rings on the fingers which can generate passive acoustic information. The sounds produced by the collision of rings of different material can be distinguished in real-time using machine learning methods. For instance, the sound produced by the collision between two metal objects will be different from the sound produced by the collision between a wood object and a metal object. These distinguishable clicking sounds can be used as additional and alternative inputs to a device to provide more options for interactions alongside the touch interface.

4.2 System Overview

We built a prototype which consists of three rings made from PLA (Polylactic Acid), plywood and metal to produce sounds. Produced sound is

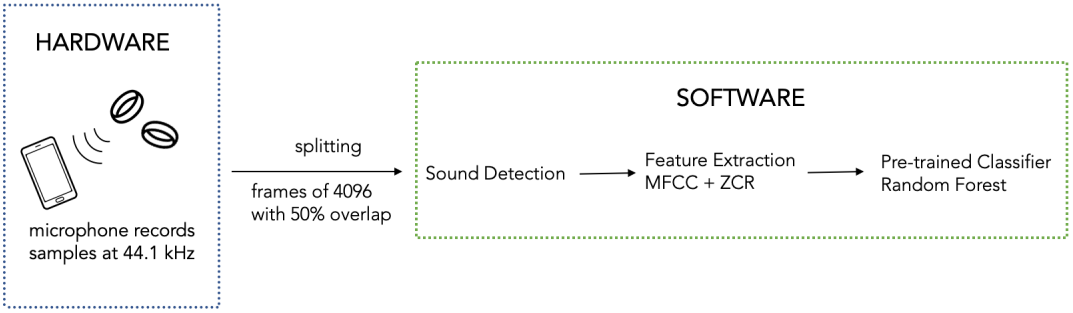

Figure 4.1: The microphone on the device keeps recording audio data. We use a sound detection technique to detect click sounds. Once click sounds are detected, audio features from this buffer will be extracted and sent to a pre-trained classifier to find the best match.

captured by an existing microphone on the mobile device. Then raw data from the buffer will be analyzed on the device.

We used an open source toolkit named TarsosDSP [41] to process the audio signal and extract features. Once a sound in a buffer is detected, we extract features of this buffer and send the feature vector to a classifier. In the classification stage, we used Random Forest classifier from the Weka Classification Library to find the best match. And the mobile device outputs a particular response based on the result.

The system is designed to run on Android devices. We leveraged the existing microphone embedded on the mobile phone to collect audio information. The architecture of the system is shown in Figure 4.1

4.3 Prototype

We 3D printed rings with platforms using PLA which can be worn on the finger flanges. Then we attached different materials to them. As figure 4.2 showed, the ring on the thumb and one of rings on the index have metal blocks attached to them. The other one on the index finger has plywood blocks attached to it. We tested with different materials to make sure that they meet two conditions. The first one is sounds created by tapping on the material are easy to capture by. The second one is sounds produced by different materials are distinguishable.

4.4 Data Collection

We designed an application running on the Android device which can record audio and add labels to it. Our system uses the Android AudioRecord class to store recently recorded audio data into a data buffer, from which features will be extracted.

To collect the training data for our model, we invited multiple participants to wear the rings and perform a few hundred clicks from each. We instructed each of the participants to make two types of sounds with the rings, which we will call sound 1 and sound 2. Sound 1 consists of clicks between two metal rings, whereas sound 2 consists of clicks between a metal ring and a plywood ring. These two individual sounds were concatenated into two separate datasets. In addition, we included a third sound dataset, called "neither", which consists of environmental noise and anything that is not sound 1 and

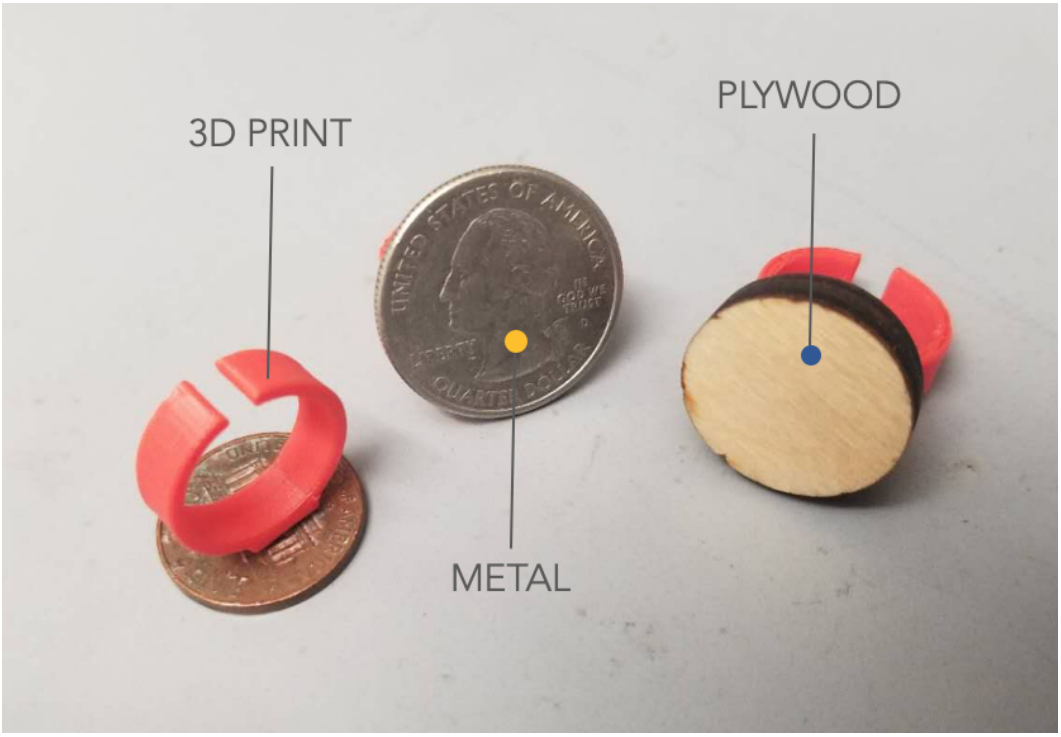

Figure 4.2: We 3D printed the body of the ring and added stick metal or plywood to it.

not sound 2. Each person was provided with the same instructions on how to wear the rings and how to properly click them. Sounds were collected in a lab environment. Participants performed each of the clicking sounds under controlled conditions, in which the most common form of noise was white noise from a fan. We also collected environmental noise for the "neither" dataset in the lab by making many different types of common sounds in everyday environments. These sounds were recorded using the embedded microphone on an Android phone. All the data, including feature vectors and class labels, is kept in an ARFF file to be used to train the model.

We used the same way to collect the test dataset. Sound 1 and sound 2 in the test dataset are from the same participants. But the training dataset and test dataset are independent to each other which means data in the test dataset has not been used to train the model.

As a result, after balancing the number of samples in each class, the training dataset has 500 samples for each and the test dataset has 70 samples for each class.

4.5 Sound Detection

While the app is open, it will access the microphone of the Android device to continuously record sound. In general, Android devices can support five different sample rates for audio recording: 8000Hz, 11025Hz, 16000Hz, 22050Hz, and 44100Hz. Recording with a high sample rate preserves more information about the original signal, but requires more time for processing.

By observing the time-domain plot of the signals, we found that click sounds last around 25ms. We selected to record sound in buffer sizes of 4096 samples, at a frequency of 44.1 kHz which is same as in the data collection stage. We found that the time of collecting 4096 samples is still slightly longer than the sum of the audio signal processing time and classification time. And recording at a frequency of 44.1 kHz gives a better result than the others. Each subsequent buffer overlaps the previous buffer by 50 percent which is 2048 samples in our system. To reduce the consumption of energy, we apply a sound detector on each buffer instead of directly extracting features. After the buffer is filled, our system will detect if there are rapid rises in energy of the signal [3]. Feature extraction will be triggered if buffers have rapid rises. This process is implemented by the percussion onset detector in TarsosDSP.

4.6 Feature Extraction

For each buffer, two primary features are extracted: MFCC and Zero-Crossing Rate.

The MFCC function in TarsosDSP is calculated with six parameters. The two required parameters are the number of samples in each frame and the sample rate of the signal. The four optional parameters are the number of cepstral coefficients, the number of mel filters, low pass filter, and high pass filter. The amount of cepstral coefficients decides the dimensionality of MFCC features vector. We used 20-dimensional MFCC feature vectors in our system. We applied a FFT to the click sound from the microphone to observe the

sound signal on the frequency domain. The resulting graphic indicated that click sound generally resides in the 3000Hz to 30kHz range. So, we set the low pass filter to be 3000Hz and high pass filter to be 30kHz.

Zero crossing rate produces a single value equal to the number of times that a signal crosses the zero-axis. This value is concatenated with the MFCC feature vector to produce the final feature vector that will be used to train the classifier.

4.7 Classification

We used Weka for Android in the classification stage. Weka for Android is the same as the classic Weka Classification Library, except without the GUI components so that it can work on Android. We tested the performance of several basic classifiers with 10-fold cross validation using the same dataset. The results in Table 4.1 indicate that Random Forest (RF) performed the best, compared with SVM (SMO in Weka) and MultilayerPerceptron (MLP), using the default hyperparameters for each model. Thus, we used our training dataset to train a random forest model and exported it as a file. This file keeps parameters of a well-trained random forest model. It can be included in an Android project directory to be used in an Android application. We performed multiple optimizations, such as limiting tree depth, number of features per tree, and number of trees to make classification as quick as possible. The intent is that each prediction can finish the process of feature extraction and classification before the next buffer is filled.

Model Comparison			
Model	Accuracy	Precision	Recall
MLP	0.933	0.933	0.933
RF	0.940	0.940	0.940
SMO	0.820	0.822	0.820

Table 4.1: We compare the performance of several basic classifiers, using the default hyperparameters for each.

4.8 Performance Optimization

A crucial aspect of real-time systems is the hard upper-bound time constraint on specific processes. In the case of our system, it's necessary to repeatedly perform sound predictions within a time interval while maintaining a standard of quality. In other words, there exists a trade-off between the amount of time for processing and the accuracy of our predictions. The goal is to maximize accuracy while keeping a bounded delay on classification time. This bound is roughly equal to the quantity of the buffer size divided by the sampling frequency, in seconds.

The three main processing steps that we focus on are getting the MFCC vector, getting the zero crossing rate value, and getting the Random Forest output using the MFCC and zero crossing rate features as input. To measure the run-time of each of these calculations, we use built-in Java function System.nanoTime() for precise timing. The aim is to consistently run all three of these steps within the delay time. Empirically, we observe that the main two points for optimization lie in MFCC and Random Forest.

In general, it takes more time to get more features for the MFCC so

it's beneficial to reduce the number of MFCC features. This motivates finding feature importance in a Random Forest model to remove unimportant features. Feature importance is measured by the decrease in node impurity weighted by the probability of reaching that node. The node probability is equal to the number of samples that reach the node, divided by the total number of samples. Put simply, a higher value signifies higher feature importance. To determine feature importance in our model, we trained a Random Forest classifier using scikit-learn library for Python, using the same dataset that we used for Weka. Its output gives normalized feature importance values that can be used to determine which features to drop to simplify the model, thus improving the model's performance in real-time. Regarding 20-feature MFCC vectors, it is considered common practice to keep only the first 13 features and drop the remaining ones. We examine the impact of various features on the overall performance of the model later in the Evaluation section.

Optimization for Random Forest focuses on tuning on three primary parameters: tree depth, number of trees, and number of features per tree. We ran a heuristic iterative algorithm shown in Algorithm 1 to optimize the values. We chose for the algorithm to run for T = 10 iterations because we observed that it had always converged after 10 iterations. Tree depth was tuned with value D = 2,...,20, as anything larger than 20 would be too slow for real-time predictions, given our buffer size of 4096 and sampling frequency of 44.1 kHz. For a similar reason, the number of trees was optimized with values N = 1,...,20 and the number of features per tree was chosen from F =

2,...,m, where m was the total number of features from MFCC and the zero crossing rate. The resultant values proved to be the best out of all of the ones obtained from an extensive search over many combinations of values for each parameter.

Algorithm 1 RF Parameter Tuning

d = tree depth
n = number of trees
f = number of features per tree
RF(d,n,f) = RF classifier accuracy trained with d, n, f
for $t = 1..T$ **do**
  $d = \arg\max_{d_i \in D} RF(d_i, n, f)$
  $n = \arg\max_{n_i \in N} RF(d, n_i, f)$
  $f = \arg\max_{f_i \in F} RF(d, n, f_i)$
end

Chapter 5

Evaluation

In this chapter, we describe the evaluation process including experimental design and experimental results.

5.1 Dataset

As described in the implementation chapter, we sampled the audio input at the highest rate (44100Hz) supported by android devices. We tested recording with a lower sampling rate (8000Hz, 11025Hz and 22050Hz). The results indicate that lowering the sampling rate will decrease the accuracy of classification.

We started with using 20 samples for each class to train the model. Based on the performance of the model, we gradually increased the number of samples in each class. When the number of samples in each class added to 500, we found that enlarging dataset did not improve the performance of the model. In addition, balancing the data in each class is helpful to build a better model.

5.2 Feature Quality Experiment

When choosing the number of dimensions for the MFCC, we experimented with 13 features, 20 features, and 40 features, all of which are commonly considered in sound recognition tasks. For each of these values, we also adjusted the Random Forest parameters accordingly when testing, using Algorithm 1, to get the best possible results. We ended up choosing 20 features for the MFCC, since it significantly outperformed the others. Regarding feature importance, we trained a Random Forest model using the scikit-learn library in Python, on the 20-feature MFCC vector and the zero crossing rate. Then, we retrieved the feature importance values using a built-in function. The results indicated that the first three features of the 20-feature MFCC were the most important, and that the zero crossing rate was the most important feature overall. When we trained a separate Random Forest model on the first three features of the MFCC and the zero crossing rate value, which will be referred to as the "simplified model", we got results that were nearly as good as the model that was trained on all the features. However, since the original model sufficiently satisfied all constraints in terms of run time and accuracy, there was no need to use the simplified model for its advantage in terms of run time.

5.3 Accuracy Experiment

For the optimizing hyperparameters of the Random Forest Classifier, we used the iterative scheme shown in Algorithm 1. As a result we chose to

	Precision	Recall
5 folds	0.925	0.925
7 folds	0.930	0.930
10 folds	0.931	0.931
test dataset	0.911	0.910

Table 5.1: We evaluate our model with cross validation and test dataset, using depth = 10, estimators = 17, and number of features = 5.

configure parameters as, number of features is 5, maximum depth is 10, and the number of trees is 17. We used 5 folds, 7 folds, and 10 folds cross validation as well as a test dataset with more than 70 samples for each class to evaluate our model. Table 5.1 shows the precision and recall values from testing. We also compared the classification results for each class since the results of sound type 1 and sound type 2 is important for us. Figure 5.1 shows the precision and recall comparison among classes. The Random Forest model in Table 4.1 is trained on 100 estimators (trees), whereas the models in Table 5.1 are each trained on 17 estimators, hence the decrease in accuracy, but increase in prediction speed.

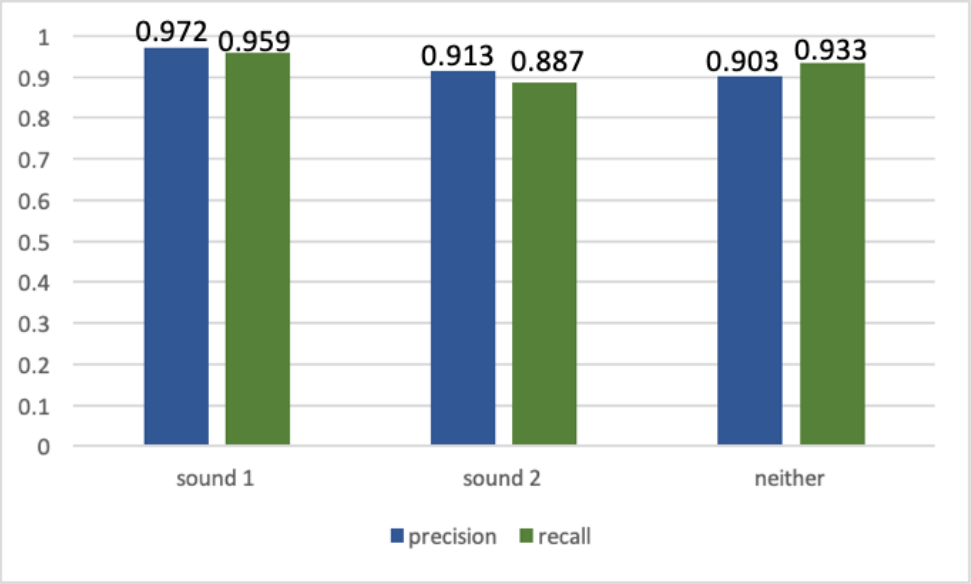

Figure 5.1: We use a test dataset to compare the precision and recall of each class.

Chapter 6

Discussion

In this chapter, we describe opportunities and challenges of our idea and system.

6.1 Application Scenarios

The final product of our system is a drawing application for Android devices. The user can change tools by tapping flange of the index finger. For example, if sound 1 is bound to the color yellow, and sound 2 is bound to the color blue, then the user can quickly swap between the two colors without needing to perform any excessive tapping with the touch screen.

Another possible application of our system, that is currently in development is for mobile games. Fast-paced mobile games have inherent limitations on the number of unique options for a user to interact with the screen in a short amount of time. Usually, a tap or a tap and drag motion are the only two quick and simple motions that can be performed. Our system remedies this issue by introducing a new array of fast options that don't even require touching the screen.

6.2 Improvements

A notable improvement could be made to the data collection stage since we only recruited two people to participate in collecting click samples. For a more versatile system, it would be best to collect data from a wider range of people to capture a larger variety of clicking sounds. In addition, the model would also benefit from data that is collected from study in the wild that is conducted in more places, rather than just limited to a university campus.

In terms of feature extraction, multiple successful works use two additional features that we did not use when training our model. These two features are spectral centroid and spectral flatness. Spectral centroid is a measure of the brightness of a sound [38]. Brightness is important for instrumental sounds, which our percussive sounds are closely related to. A higher value of spectral centroid indicates a brighter sound. Spectral flatness is closely related to spectral centroid, and is a measure of the quality of the tone of a sound. Spectral flatness excels at distinguishing between tones and more percussive sounds. Since our sounds consist of a ringing metal click and a short wooden click, spectral flatness would be a good consideration as another feature.

For classification, a neural network could possibly be a better alternative to the Random Forest model. We trained a neural network on the data and achieved 99 percent accuracy. However, we were unable to run it in real-time. In the future, we will continue to put effort into making the neural network faster, while maintaining its level of accuracy.

6.3 Limitations

Different tapping habits of users may result in subtle differences in sound signals. For instance, the duration of the sound and the intensity of the sound. To eliminate this difference, in the data collection stage, we recruited multiple participants to click instead of clicking by one person. The training model was build using the data provided by these participants. However, when new users try to use this system, their click samples are not included in the training dataset which may cause a decrease in recognition accuracy. This indicates that training dataset may need to include various click samples to make the model work for more people.

We have another sound class besides target sound classes. Neither sound consists of environmental noise which is hard to define. In our case, all the impulsive sounds except click type 1 and click type 2 are regarded as environment noise. From the classification results, we found that our system often predicts environmental sound to be sound 1 or sound 2. Therefore, we think that it is better to include as many as possible various environmental sounds from daily life. In addition, there is a great possibility that two metal objects hit each other in the environment. Our system regards these sound as our target sound. For example, a coin falling on a metal platform is easy to recognize as click type 1, which is produced by two metal blocks.

6.4 Future Works

From the perspective of material selection, materials in nature are not limited to metal and plywood. There are still some other kinds of material for us to explore. For example, clicking on acrylic or plastic can also create unique sounds which can be used in our situation. Besides, the structure of objects also affect the timbre of sounds. Making the block on the ring hollow or having a specific pattern inside can create more kinds of sounds. In fact, making longer lasting, more tone-like sounds could be more distinguishing, especially to the human ear.

At the moment, we only tested tapping gesture to produce sounds. In fact, sound can be produced in many different ways. For example, swiping up or down on a uneven surface can produce a short piece of audio with certain characteristics. This type of sound can be extracted from environmental noise. And they are special enough so that they will not be easily confused with environmental noise because it rarely appears in daily life.

Lastly, a possibility is to combine two types of sounds as one action. Inspired from double clicking with a mouse, double tapping on the material can trigger different responses from single tapping. We treat two taps as double-tap if the interval between two taps is short enough.

Chapter 7

Conclusion

Traditional interaction methods with mobile devices rely on touching screens using fingers. When the primary finger is engaged in a touch event, other fingers are regarded as idle fingers. Considering that human fingers are dexterous, we expect leveraging remaining fingers to extend ways of input. We designed an input technology that recognizes sounds produced by two rings on fingers. Users can make different click sounds by wearing rings made from different materials. Our system makes a particular response by recognizing these sounds. Through this method, we enhanced the traditional touch-based interaction.

We explored the possibility to recognize passive finger sounds using machine learning methods. And we developed an application deployed on Android to demonstrate it. The entire process including capture sounds, detecting sounds, extracting features and recognition runs directly on a mobile device without Internet connection in real-time.

Appendices

Appendix A

Additional Tables and Figures

As mentioned in the implementation chapter, we ran a heuristic iterative algorithm to optimize the values. We plotted the results for making decisions. Figure A.1, figure A.2 and figure A.3 show the running results.

	Extract MFCC	Extract ZCR	Classification
Min Time Cost (ms)	0.100	0.080	115.510
Max Time Cost (ms)	0.954	0.259	185.880

Table A.1: We calculated the time consumption of each stage from 15 results using built-in Java function System.nanoTime(). From the table, we can see that time is mainly consumed in the classification stage. Extracting features does not cost much time. This indicates that pruning the model to make it light is important.

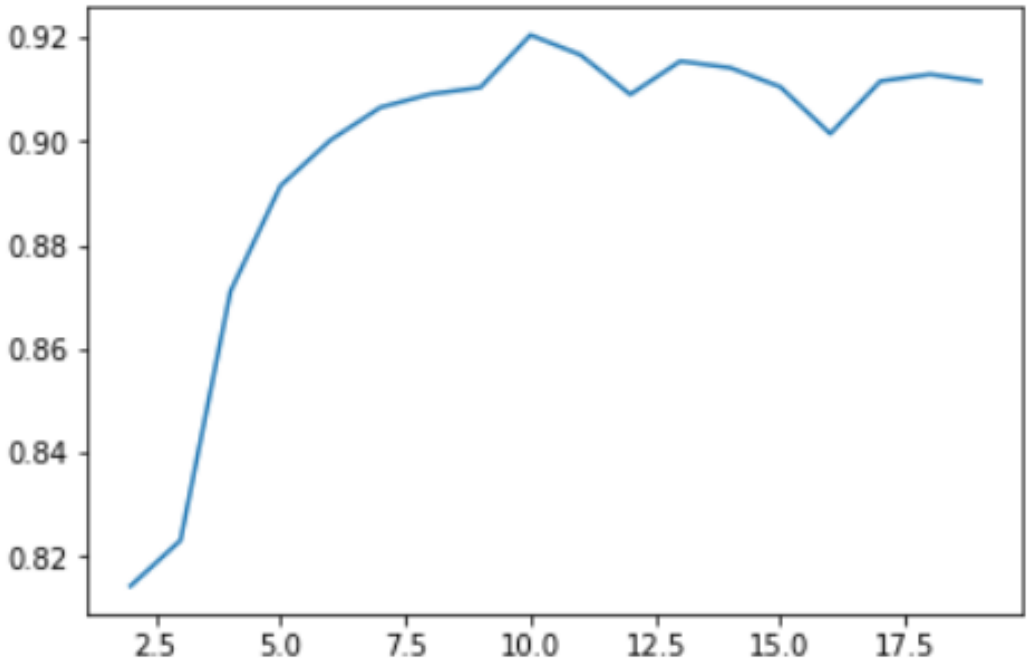

Figure A.1: We tested with parameters as number of features is 5, number of trees is 17. The x-axis is maximum depth which is set from 2 to 20. The y-axis is the accuracy value.

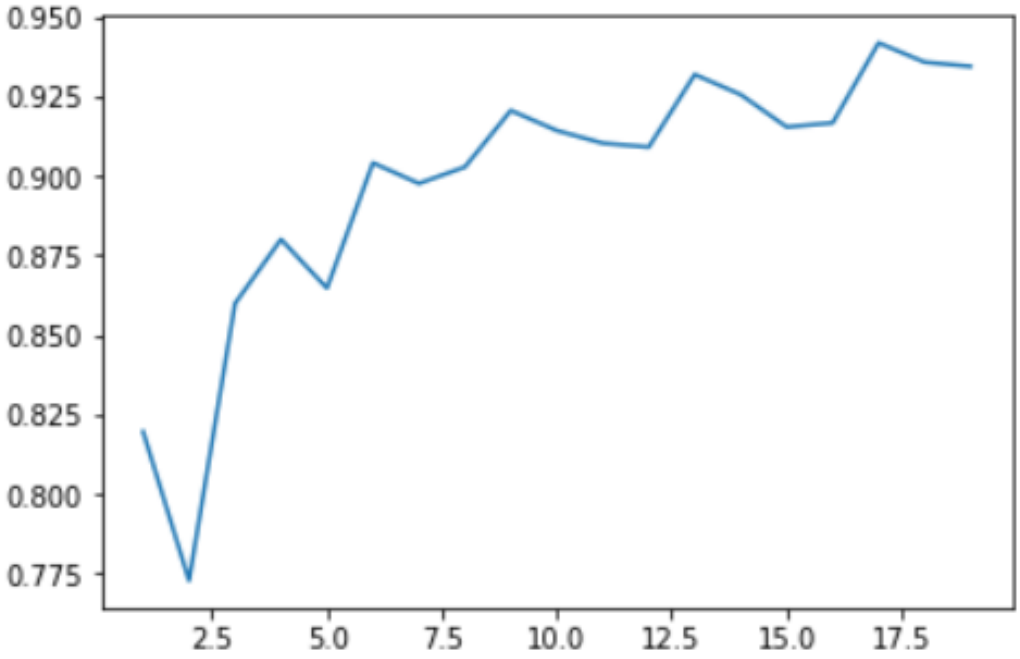

Figure A.2: We tested with parameters as maximum depth is 10, number of features is 5. The x-axis is the number of trees which is set from 1 to 20. The y-axis is the accuracy value.

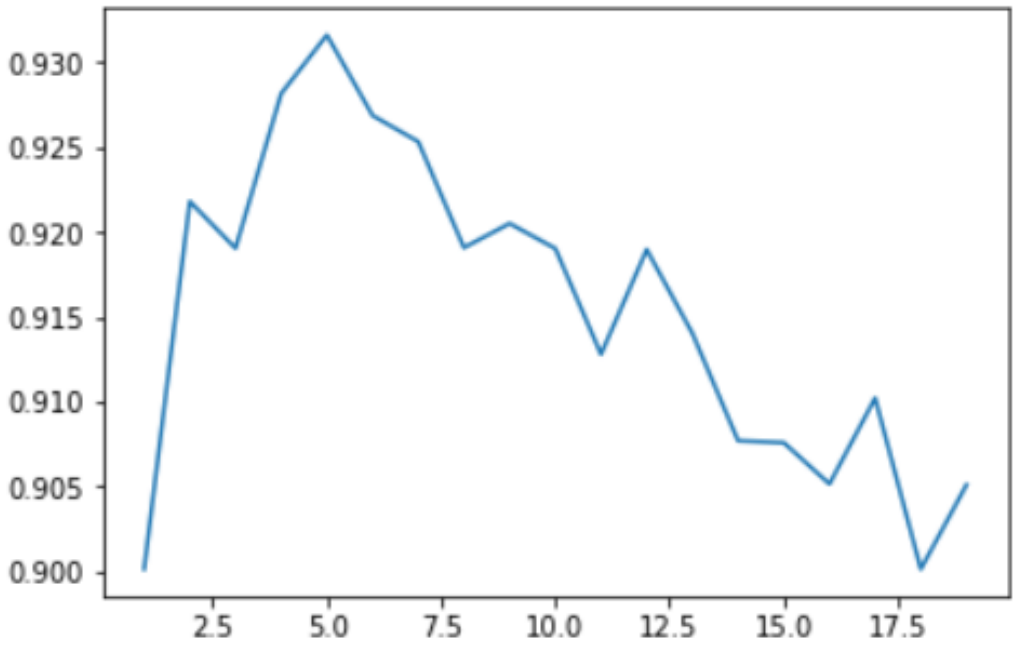

Figure A.3: We tested with parameters as maximum depth is 10, number of trees is 17. The x-axis is the number of features which is set from 1 to 20. The y-axis is the accuracy value.

Appendix B

Experimental Results from Weka

As mentioned in the evaluation chapter, we used 5 folds, 7 folds and 10 folds cross validation as well as a test dataset to evaluate our model's performance. The parameters of the random forest model are bagging with 17 iterations, using 5 features with depth of 10. We list the summary page from weka for each method. s1 represents sound type 1, s2 represents sound type 2 and neither represents environmental sound. The model often misjudges noise sound to be sound type 1 or sound type 2. This indicates that adding variety samples to the nether class is helpful.

```
=== Summary ===

Correctly Classified Instances        1396               92.5116 %
Incorrectly Classified Instances       113                7.4884 %
Kappa statistic                          0.8877
Mean absolute error                      0.1378
Root mean squared error                  0.2199
Relative absolute error                 31.0091 %
Root relative squared error             46.6561 %
Total Number of Instances             1509

=== Detailed Accuracy By Class ===

                 TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
                 0.940    0.031    0.938      0.940    0.939      0.909    0.987     0.980     s1
                 0.946    0.044    0.916      0.946    0.931      0.895    0.987     0.980     s2
                 0.889    0.038    0.922      0.889    0.905      0.859    0.978     0.955     neither
Weighted Avg.    0.925    0.037    0.925      0.925    0.925      0.888    0.984     0.972

=== Confusion Matrix ===

   a   b   c   <-- classified as
 471   9  21 |   a = s1
  10 477  17 |   b = s2
  21  35 448 |   c = neither
```

Figure B.1: The summary of evaluation using 5 fold cross validation.

```
=== Summary ===

Correctly Classified Instances        1404               93.0417 %
Incorrectly Classified Instances       105                6.9583 %
Kappa statistic                          0.8956
Mean absolute error                      0.1366
Root mean squared error                  0.2177
Relative absolute error                 30.7359 %
Root relative squared error             46.185  %
Total Number of Instances             1509

=== Detailed Accuracy By Class ===

                 TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
                 0.954    0.039    0.925      0.954    0.939      0.908    0.988     0.980     s1
                 0.940    0.032    0.937      0.940    0.939      0.908    0.985     0.973     s2
                 0.897    0.034    0.930      0.897    0.913      0.871    0.980     0.964     neither
Weighted Avg.    0.930    0.035    0.930      0.930    0.930      0.896    0.985     0.972

=== Confusion Matrix ===

   a   b   c   <-- classified as
 478   9  14 |   a = s1
  10 474  20 |   b = s2
  29  23 452 |   c = neither
```

Figure B.2: The summary of evaluation using 7 fold cross validation.

```
=== Summary ===

Correctly Classified Instances        1405               93.108  %
Incorrectly Classified Instances       104                6.892  %
Kappa statistic                          0.8966
Mean absolute error                      0.1289
Root mean squared error                  0.2121
Relative absolute error                 28.998  %
Root relative squared error             45.0028 %
Total Number of Instances             1509

=== Detailed Accuracy By Class ===

                 TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
                 0.946    0.033    0.935      0.946    0.940      0.911    0.989     0.984     s1
                 0.954    0.037    0.929      0.954    0.941      0.911    0.986     0.978     s2
                 0.893    0.034    0.930      0.893    0.911      0.868    0.981     0.960     neither
Weighted Avg.    0.931    0.034    0.931      0.931    0.931      0.897    0.985     0.974

=== Confusion Matrix ===

   a   b   c   <-- classified as
 474   9  18 |   a = s1
   7 481  16 |   b = s2
  26  28 450 |   c = neither
```

Figure B.3: The summary of evaluation using 10 fold cross validation.

```
=== Summary ===

Correctly Classified Instances         213               91.0256 %
Incorrectly Classified Instances        21                8.9744 %
Kappa statistic                          0.8645
Mean absolute error                      0.1629
Root mean squared error                  0.246
Relative absolute error                 36.6525 %
Root relative squared error             52.1942 %
Total Number of Instances              234

=== Detailed Accuracy By Class ===

                 TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
                 0.959    0.043    0.909      0.959    0.933      0.903    0.985     0.978     s1
                 0.873    0.025    0.939      0.873    0.905      0.867    0.967     0.942     s2
                 0.900    0.069    0.890      0.900    0.895      0.829    0.968     0.948     neither
Weighted Avg.    0.910    0.048    0.911      0.910    0.910      0.863    0.973     0.956

=== Confusion Matrix ===

  a  b  c   <-- classified as
 70  0  3 |  a = s1
  2 62  7 |  b = s2
  5  4 81 |  c = neither
```

Figure B.4: The summary of evaluation using a test dataset.

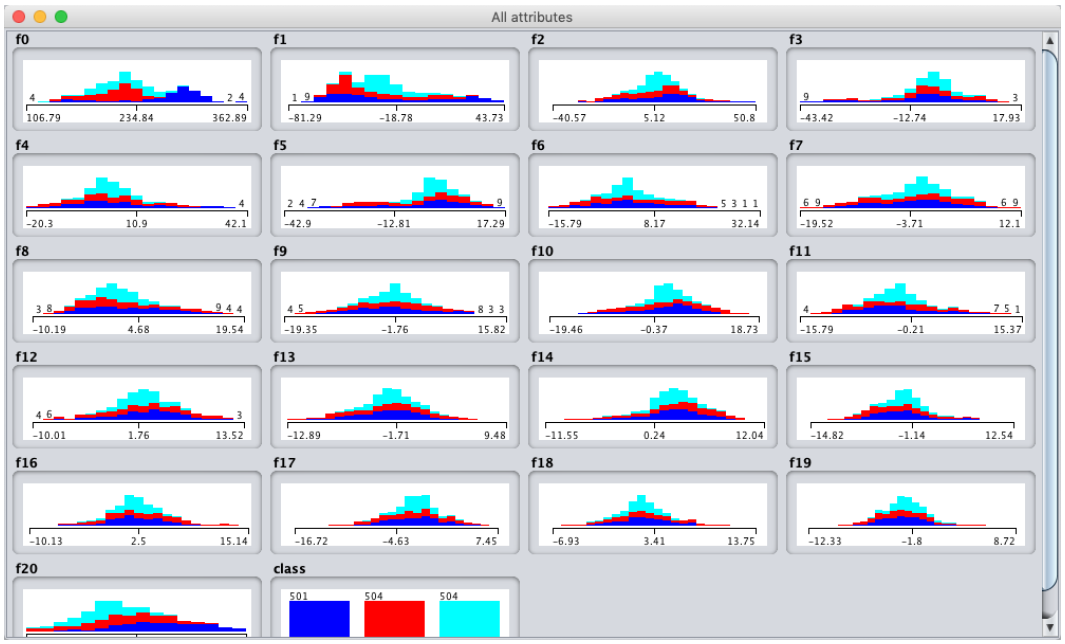

Figure B.5: The visualization of the distribution of 21 features. f0 - f19 are 20 MFCC features. f20 is zero crossing rate

Bibliography

[1] Brian Amento, Will Hill, and Loren Terveen. The sound of one hand: A wrist-mounted bio-acoustic fingertip gesture interface. In *CHI'02 Extended Abstracts on Human Factors in Computing Systems*, pages 724–725. ACM, 2002.

[2] Daniel Ashbrook, Patrick Baudisch, and Sean White. Nenya: subtle and eyes-free mobile input with a magnetically-tracked finger ring. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 2043–2046. ACM, 2011.

[3] Dan Barry, Derry Fitzgerald, Eugene Coyle, and Bob Lawlor. Drum source separation using percussive feature detection and spectral modulation. 2005.

[4] Robin Biondi, Gareth Dys, Gilles Ferone, Thibault Renard, and Morgan Zysman. Low cost real time robust identification of impulsive signals. *International Journal of Computer, Electrical, Automation, Control and Information Engineering*, 8(9):1653–1656, 2014.

[5] Sebastian Boring, David Ledo, Xiang'Anthony' Chen, Nicolai Marquardt, Anthony Tang, and Saul Greenberg. The fat thumb: using the thumb's contact size for single-handed mobile interaction. In *Proceedings of the*

14th international conference on Human-computer interaction with mobile devices and services, pages 39–48. ACM, 2012.

[6] William Brent. Cepstral analysis tools for percussive timbre identification. In *Proceedings of the 3rd International Pure Data Convention, Sao Paulo, Brazil*. sn, 2009.

[7] Alex Butler, Shahram Izadi, and Steve Hodges. Sidesight: multi-touch interaction around small devices. In *Proceedings of the 21st annual ACM symposium on User interface software and technology*, pages 201–204. ACM, 2008.

[8] Liwei Chan, Yi-Ling Chen, Chi-Hao Hsieh, Rong-Hao Liang, and Bing-Yu Chen. Cyclopsring: Enabling whole-hand and context-aware interactions through a fisheye ring. In *Proceedings of the 28th Annual ACM Symposium on User Interface Software & Technology*, pages 549–556. ACM, 2015.

[9] Jianfeng Chen, Alvin Harvey Kam, Jianmin Zhang, Ning Liu, and Louis Shue. Bathroom activity monitoring based on sound. In *International Conference on Pervasive Computing*, pages 47–61. Springer, 2005.

[10] Ke-Yu Chen, Kent Lyons, Sean White, and Shwetak Patel. utrack: 3d input using two magnetic sensors. In *Proceedings of the 26th annual ACM symposium on User interface software and technology*, pages 237–244. ACM, 2013.

[11] Ke-Yu Chen, Shwetak N Patel, and Sean Keller. Finexus: Tracking precise motions of multiple fingertips using magnetic sensing. In *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems*, pages 1504–1514. ACM, 2016.

[12] Xiang'Anthony' Chen, Julia Schwarz, Chris Harrison, Jennifer Mankoff, and Scott E Hudson. Air+ touch: interweaving touch & in-air gestures. In *Proceedings of the 27th annual ACM symposium on User interface software and technology*, pages 519–525. ACM, 2014.

[13] Selina Chu, Shrikanth Narayanan, and C-C Jay Kuo. Environmental sound recognition with time–frequency audio features. *IEEE Transactions on Audio, Speech, and Language Processing*, 17(6):1142–1158, 2009.

[14] Michael Cowling and Renate Sitte. Comparison of techniques for environmental sound recognition. *Pattern recognition letters*, 24(15):2895–2907, 2003.

[15] Namrata Dave. Feature extraction methods lpc, plp and mfcc in speech recognition. *International journal for advance research in engineering and technology*, 1(6):1–4, 2013.

[16] Artem Dementyev and Joseph A Paradiso. Wristflex: low-power gesture input with wrist-worn pressure sensors. In *Proceedings of the 27th annual ACM symposium on User interface software and technology*, pages 161–166. ACM, 2014.

[17] Alain Dufaux, Laurent Besacier, Michael Ansorge, and Fausto Pellandini. Automatic sound detection and recognition for noisy environment. In *2000 10th European Signal Processing Conference*, pages 1–4. IEEE, 2000.

[18] Tiantian Feng, Amrutha Nadarajan, Colin Vaz, Brandon Booth, and Shrikanth Narayanan. Tiles audio recorder: an unobtrusive wearable solution to track audio activity. In *Proceedings of the 4th ACM Workshop on Wearable Systems and Applications*, pages 33–38. ACM, 2018.

[19] Aurélien Géron. *Hands-on machine learning with Scikit-Learn and TensorFlow: concepts, tools, and techniques to build intelligent systems.* " O'Reilly Media, Inc.", 2017.

[20] Sarthak Ghosh, Hyeong Cheol Kim, Yang Cao, Arne Wessels, Simon T Perrault, and Shengdong Zhao. Ringteraction: Coordinated thumb-index interaction using a ring. In *Proceedings of the 2016 CHI Conference Extended Abstracts on Human Factors in Computing Systems*, pages 2640–2647. ACM, 2016.

[21] Fabien Gouyon, François Pachet, Olivier Delerue, et al. On the use of zero-crossing rate for an application of classification of percussive sounds. In *Proceedings of the COST G-6 conference on Digital Audio Effects (DAFX-00), Verona, Italy*, 2000.

[22] Sidhant Gupta, Daniel Morris, Shwetak Patel, and Desney Tan. Soundwave: using the doppler effect to sense gestures. In *Proceedings of the*

SIGCHI Conference on Human Factors in Computing Systems, pages 1911–1914. ACM, 2012.

[23] Chris Harrison, Hrvoje Benko, and Andrew D Wilson. Omnitouch: wearable multitouch interaction everywhere. In *Proceedings of the 24th annual ACM symposium on User interface software and technology*, pages 441–450. ACM, 2011.

[24] Chris Harrison, Julia Schwarz, and Scott E Hudson. Tapsense: enhancing finger interaction on touch surfaces. In *Proceedings of the 24th annual ACM symposium on User interface software and technology*, pages 627–636. ACM, 2011.

[25] Chris Harrison, Desney Tan, and Dan Morris. Skinput: appropriating the body as an input surface. In *Proceedings of the SIGCHI conference on human factors in computing systems*, pages 453–462. ACM, 2010.

[26] David Kim, Otmar Hilliges, Shahram Izadi, Alex D Butler, Jiawen Chen, Iason Oikonomidis, and Patrick Olivier. Digits: freehand 3d interactions anywhere using a wrist-worn gloveless sensor. In *Proceedings of the 25th annual ACM symposium on User interface software and technology*, pages 167–176. ACM, 2012.

[27] Gierad Laput, Karan Ahuja, Mayank Goel, and Chris Harrison. Ubicoustics: Plug-and-play acoustic activity recognition. In *The 31st Annual ACM Symposium on User Interface Software and Technology*, pages 213–224. ACM, 2018.

[28] Gierad Laput, Robert Xiao, Xiang'Anthony' Chen, Scott E Hudson, and Chris Harrison. Skin buttons: cheap, small, low-powered and clickable fixed-icon laser projectors. In *Proceedings of the 27th annual ACM symposium on User interface software and technology*, pages 389–394. ACM, 2014.

[29] Gierad Laput, Robert Xiao, and Chris Harrison. Viband: High-fidelity bio-acoustic sensing using commodity smartwatch accelerometers. In *Proceedings of the 29th Annual Symposium on User Interface Software and Technology*, pages 321–333. ACM, 2016.

[30] Zhu Le-Qing. Insect sound recognition based on mfcc and pnn. In *2011 International Conference on Multimedia and Signal Processing*, volume 2, pages 42–46. IEEE, 2011.

[31] Hyunchul Lim, Jungmin Chung, Changhoon Oh, SoHyun Park, Joonhwan Lee, and Bongwon Suh. Touch+ finger: Extending touch-based user interface capabilities with idle finger gestures in the air. In *The 31st Annual ACM Symposium on User Interface Software and Technology*, pages 335–346. ACM, 2018.

[32] Héctor Lozano, Inmaculada Hernáez, Javier Camarena, Ibai Díez, and Eva Navas. A real-time sound recognition system in an assisted environment. In *International Conference on Computers for Handicapped Persons*, pages 385–391. Springer, 2012.

[33] Lindasalwa Muda, Mumtaj Begam, and Irraivan Elamvazuthi. Voice recognition algorithms using mel frequency cepstral coefficient (mfcc) and dynamic time warping (dtw) techniques. *arXiv preprint arXiv:1003.4083*, 2010.

[34] Adiyan Mujibiya, Xiang Cao, Desney S Tan, Dan Morris, Shwetak N Patel, and Jun Rekimoto. The sound of touch: on-body touch and gesture sensing based on transdermal ultrasound propagation. In *Proceedings of the 2013 ACM international conference on Interactive tabletops and surfaces*, pages 189–198. ACM, 2013.

[35] K Sri Rama Murty and Bayya Yegnanarayana. Combining evidence from residual phase and mfcc features for speaker recognition. *IEEE signal processing letters*, 13(1):52–55, 2006.

[36] Rajalakshmi Nandakumar, Vikram Iyer, Desney Tan, and Shyamnath Gollakota. Fingerio: Using active sonar for fine-grained finger tracking. In *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems*, pages 1515–1525. ACM, 2016.

[37] Masa Ogata, Yuta Sugiura, Hirotaka Osawa, and Michita Imai. iring: intelligent ring using infrared reflection. In *Proceedings of the 25th annual ACM symposium on User interface software and technology*, pages 131–136. ACM, 2012.

[38] Angelos Pillos, Khalid Alghamidi, Noura Alzamel, Veselin Pavlov, and Swetha Machanavajhala. A real-time environmental sound recognition

system for the android os. *Proceedings of Detection and Classification of Acoustic Scenes and Events*, 2016.

[39] Lawrence R Rabiner, Biing-Hwang Juang, and Janet C Rutledge. *Fundamentals of speech recognition*, volume 14. PTR Prentice Hall Englewood Cliffs, 1993.

[40] Mirco Rossi, Sebastian Feese, Oliver Amft, Nils Braune, Sandro Martis, and Gerhard Tröster. Ambientsense: A real-time ambient sound recognition system for smartphones. In *2013 IEEE International Conference on Pervasive Computing and Communications Workshops (PERCOM Workshops)*, pages 230–235. IEEE, 2013.

[41] Joren Six, Olmo Cornelis, and Marc Leman. TarsosDSP, a Real-Time Audio Processing Framework in Java. In *Proceedings of the 53rd AES Conference (AES 53rd)*, 2014.

[42] Jie Song, Gábor Sörös, Fabrizio Pece, Sean Ryan Fanello, Shahram Izadi, Cem Keskin, and Otmar Hilliges. In-air gestures around unmodified mobile devices. In *Proceedings of the 27th annual ACM symposium on User interface software and technology*, pages 319–329. ACM, 2014.

[43] Feng Wang, Xiang Cao, Xiangshi Ren, and Pourang Irani. Detecting and leveraging finger orientation for interaction with direct-touch surfaces. In *Proceedings of the 22nd annual ACM symposium on User interface software and technology*, pages 23–32. ACM, 2009.

[44] Jia-Ching Wang, Hsiao-Ping Lee, Jhing-Fa Wang, and Cai-Bei Lin. Robust environmental sound recognition for home automation. *IEEE Transactions on Automation Science and Engineering*, 5(1):25–31, 2008.

[45] Hongyi Wen, Julian Ramos Rojas, and Anind K Dey. Serendipity: Finger gesture recognition using an off-the-shelf smartwatch. In *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems*, pages 3847–3851. ACM, 2016.

[46] Ian H Witten, Eibe Frank, Mark A Hall, and Christopher J Pal. *Data Mining: Practical machine learning tools and techniques*. Morgan Kaufmann, 2016.

[47] Robert Xiao, Teng Cao, Ning Guo, Jun Zhuo, Yang Zhang, and Chris Harrison. Lumiwatch: On-arm projected graphics and touch input. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*, page 95. ACM, 2018.

[48] Sang Ho Yoon, Yunbo Zhang, Ke Huo, and Karthik Ramani. Tring: Instant and customizable interactions with objects using an embedded magnet and a finger-worn device. In *Proceedings of the 29th Annual Symposium on User Interface Software and Technology*, pages 169–181. ACM, 2016.

[49] Cheng Zhang, Anandghan Waghmare, Pranav Kundra, Yiming Pu, Scott Gilliland, Thomas Ploetz, Thad E Starner, Omer T Inan, and Gregory D

Abowd. Fingersound: Recognizing unistroke thumb gestures using a ring. *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies*, 1(3):120, 2017.

[50] Cheng Zhang, Xiaoxuan Wang, Anandghan Waghmare, Sumeet Jain, Thomas Ploetz, Omer T Inan, Thad E Starner, and Gregory D Abowd. Fingorbits: interaction with wearables using synchronized thumb movements. In *Proceedings of the 2017 ACM International Symposium on Wearable Computers*, pages 62–65. ACM, 2017.

[51] Cheng Zhang, Qiuyue Xue, Anandghan Waghmare, Ruichen Meng, Sumeet Jain, Yizeng Han, Xinyu Li, Kenneth Cunefare, Thomas Ploetz, Thad Starner, et al. Fingerping: Recognizing fine-grained hand poses using active acoustic on-body sensing. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*, page 437. ACM, 2018.

[52] Yang Zhang, Junhan Zhou, Gierad Laput, and Chris Harrison. Skintrack: Using the body as an electrical waveguide for continuous finger tracking on the skin. In *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems*, pages 1491–1503. ACM, 2016.

[53] Chen Zhao, Ke-Yu Chen, Md Tanvir Islam Aumi, Shwetak Patel, and Matthew S Reynolds. Sideswipe: detecting in-air gestures around mobile devices using actual gsm signal. In *Proceedings of the 27th annual ACM symposium on User interface software and technology*, pages 527–534. ACM, 2014.