Toolkit For Gesture Classification Through Acoustic Sensing

Pedro Miguel Gato Soldado

Thesis to obtain the Master of Science Degree in

Information Systems and Computer Engineering

Supervisor Prof. Joaquim Armando Pires Jorge

Examination Committee

Chairperson Prof. Miguel Nuno Dias Alves Pupo Correia

Supervisor Prof. Joaquim Armando Pires Jorge

Member of the Committee Prof. Carlos Alberto Pacheco Dos Anjos Duarte

November 2015

Contents

List of Figures

Abstract

The interaction with touch displays has become increasingly popular with the growth of the mobile devices market. Most people already own at least one of these devices, and use them regularly. By extension, touch displays, and all applications that make use of this technology have become quite popular, available by the thousands nowadays. However, these applications don't make full use of a user gesture's potential, focusing mainly on hand's position and shape. This work proposes a new approach to identify and classify gestures with different acoustic signatures.

To tackle this problem, we proposed an approach to gesture classification through the sound produced by interaction on a surface. This approach is provided as a development toolkit, to integrate these features into applications, while freeing the developer from the need to understand and implement complex classification and audio processing algorithms.

We detail the components of the toolkit's architecture, and describe the main processes it implements.

To explore and evaluate this approach, a set of applications was developed, that use the toolkit as a mean to improve interaction and map user's gestures to concrete actions.

The main outcome of this work is a toolkit built to simplify the development of applications that make full use of a user's gestures, and that allows an expedite mapping between the application's features and an easily distinguishable gesture.

Keywords: Acoustic sensing, toolkit development, gesture classification, audio processing, interaction, application development

Chapter 1

Introduction

1.1 Motivation

Interacting with touch displays is something that became very usual in our daily life. Most people use devices (smartphones, tablets,...) equipped with these displays everyday, sometimes even as a part of their working habits.

Everyone knows how to use these displays, and many applications have been developed, that make use of all the potential they have, like multiple finger input.

However, most of these applications do not make use of a user **gesture's potential**, focusing mainly on hand's position and shape. This leads to a loss in expressive power, specially on more complex interfaces.

There are many other acoustic dimensions that can be taken into account and explored, such as intensity and timbre, allowing the differentiation between simple gestures applied with different body parts.

This process has been called **acoustic sensing** in several works. The whole process can be relatively complex, as it includes (among others): sound capture, signal analysis, sampling, and matching against a database of known gestures. The analysis done is composed mainly by machine learning algorithms, which may not be easy for every developer to understand and use.

Also, there are currently no toolkits that have all these features readily available for use, without the need for extensive adaptation and configuration, making the reuse and adaptation of the available toolkits or the development of new ones quite difficult and time-consuming.

It is then easy to understand why most developers do not see the added value of this kind of features, as it takes much time of the development process, possibly essential to other main activities. Moreover, even by reusing some of the existent solutions available to allow gesture classification, some problems still remain:

- Further configuration is needed in some cases, such as gesture training, or to adjust the already developed application to integrate the new features;

- Default touches, when training is not available, may not be useful or appropriate to the interface's available actions.

This work will try to solve these problems by providing a simple yet powerful toolkit that includes acoustic sensing features (to distinguish users' gestures) on its core, while making it easy to be used and integrated on the developing of applications. The focus will be to provide these features to devices equipped with touch displays, in particular mobile devices (tablets and smartphones) and interactive touch tabletop surfaces.

1.2 Research Questions

The limitations and problems presented in the previous section represent a strong motivation to improve the work done on acoustic sensing and gesture classification through sound input produced on touch displays.

This main challenge can be summarized in the following question:

> *"How to simplify developing applications that use sound produced by touch input to distinguish and classify users' gestures?"*

The answer to this question is not trivial. To allow a simple development of these applications some requirements must be met:

- Additional development and integration must be as simple as possible, to not harm the application's development cycle;
- The tool must be readily available and well documented;
- The developer must have some degree of flexibility when using the tool (trainable gestures for example).

Some of these aspects might not be easy to ensure, but must be guaranteed throughout all the work to provide successful results.

After answering the previous question, related with the development process, a new question arises:

> *"How can this approach improve usability of applications by extending the interaction design space?"*

After the successful development of an application that uses our approach, there's still an important analysis to be made: how well is the developed application exercised by end users? If no improvement is observed where interaction is concerned, the final user will not be satisfied, and therefore the developer may not consider our solution as a serious contribution to application development. Thus this aspect must also be taken into account and evaluated.

1.3 Objectives

The main objective of this work is to provide a simple way to develop applications that make use of the sound produced by a user's gesture on a touch surface.

We will start by studying the related work on the areas of acoustic sensing and gesture classification, and also analyze the characteristics of the surfaces we consider on our approach: touch surfaces, both on mobile devices, and tabletop surfaces.

It is also important to study some alternatives for development of toolkits of this type, such as previous proposed solutions and their main differences, to understand possible advantages we can leverage on, and analyse some of the limitations they may have.

Then we will propose an approach to the first question presented on the previous section, in the form of a toolkit, that will help application developers use acoustic sensing and gesture classification features to improve user interaction with the application. Using our approach, a developer will not have to worry about the dynamics of acoustic sensing, or with any sound analysis and gesture classification. The idea is, by using the toolkit and with only a few lines of code, provide developers with a set of features ready to apply to the application's interface.

Finally, to evaluate the interaction and as a proof of concept, some prototype applications will be developed to showcase the toolkit's features and capabilities, and user tests will be executed using these applications.

1.4 Contributions

The research scope of this work is related to the development of new techniques to explore the interaction space on touch displays.

The main contribution of this thesis will be a set of toolkits (one version for each device type, mobile and tabletop surfaces) to allow and simply the development of applications that can leverage on acoustic sensing properties to expand the interaction on interfaces using simple but meaningful gestures.

Other than this primary contribution, in this work we will also produce:

- A survey of the state of the art on acoustic sensing tools and on the development of toolkits that are similar to our main contribution;
- Prototype applications developed using our toolkit;
- Results of all tests executed on our approach, such as gesture classification tests, toolkit performance tests, and user tests to the prototype applications.

The applications developed will illustrate the use of our toolkits to improve interaction. Two applications are going to be developed: a paint application, a simple painting application that allows the use of different gestures to activate features and change between them (using the mobile version of the toolkit), and a picture manager application, an application that provides features for managing collections of pictures (using the touch tabletop version).

1.5 Document Structure

The remainder of this document is organized as follows.

First, Chapter 2 describes the related work, by first giving a notion of what acoustic sensing is, as well as some approaches that explored it. This chapter also explores the characteristics of the devices for which the toolkits are going to be developed: mobile devices, and tabletop surfaces.

Then, Chapter 3 describes the proposed approach, focusing on the architecture design, and the main decisions that were made.

Finally, Chapter 4 presents all evaluation work done on the approach. The focus is to present the main results and methodologies used to obtain them, and the conclusions obtained.

Chapter 2

Related Work

In this chapter it is firstly introduced the concept of acoustic sensing, the characteristics that can be retrieved with it, and the main techniques used to achieve it.

Next, some of the approaches to acoustic sensing are analyzed, in particular their objectives, advantages and/or limitations.

Then, it is also important to study the devices that the approach is focused on: mobile devices and tabletop surfaces.

Finally, a brief assessment of some of the toolkits already available that enable the incorporation of gestures on applications is presented, to better understand what are the main features that can be provided, which approaches were used, and their limitations.

Before this study, there are some considerations that must be taken into account.

The first important concept to have in mind is the difference between touch and gesture.

A **touch**, on a wider sense, is the most simple action made by the user when interacting with a surface. A touch on a surface can be interpreted (e.g. capacitive displays retrieve information from the position of each touch).

A **gesture** however, is an action or sequence of actions that may convey meaning to the surface (i.e. to the system that is using that surface as interface), In this work, a gesture is considered as the association between a touch or set of touches, and the sound produced.

2.1 What is Acoustic Sensing?

The signal captured by acoustic sensors contains lots of information that can be used on the development of tangible acoustic interfaces [CL13]. These interfaces work based on the principle that when an object is touched and manipulated, its properties are altered. In particular, the way it resonates (or vibrates) when it is touched varies depending on how and where it is touched. The vibrations it produces can be captured and analyzed to infer information about how interaction with the object is being carried.

This technique is called **acoustic sensing**.

Depending on the approach used to implement acoustic sensing, and on the purpose of the work, the following information (examples) can be retrieved:

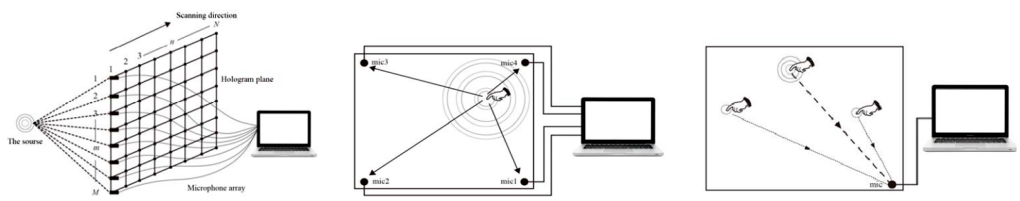
- Touch position on the object surface;
- Identification of the object used to interact with the surface;
- Estimation of the impact intensity exerted on the object;
- Distinction between continuous and discrete touches.

According to [CL13; RDS05], there are three major techniques (using acoustic sensing) that can be used to develop tangible acoustic interfaces: acoustic holography, time delay of arrival and location pattern matching (or time reversal).

Acoustic Holography is based on previous work done on the field of holography, a technique that allows the reconstruction of three-dimensional images. This technique is based on the principle of interference, as a hologram is produced by capturing the interference pattern between two or more beams of light (laser light). This idea has been explored as a technique for acoustic sensing, called acoustic holography. On this technique, as illustrated on Figure 2.1a, sound waves are recorded by microphones and processed by computer to reconstruct the sound field generated where object (or touch) position can be determined.

The **Time Delay of Arrival (TDOA)** technique is widely used for radar and sonar navigation and localization, but it can also be used to develop acoustic interfaces. When a surface is touched, vibrations are generated on the object surface. These vibrations are captured by sensors (e.g. microphones), and are later analyzed and compared. By knowing the propagation velocity of sound on the material, it is possible to calculate the distance between sensors, and determine the touch position. The process is illustrated on Figure 2.1b.

Finally, the **Location Pattern Matching (LPM)** technique, also called Time Reversal [RDS05], uses a single sensor to capture the sound wave produced by a touch on the objects surface. This sound wave contains information that can be mapped to a specific position. Every new touch on the surface is then compared to a database of saved points. Although less instrumentation is needed, performance is slow because each interaction must be matched against the database of saved positions.



(a) Acoustic Holography Diagram

(b) Time Delay of Arrival Diagram

(c) Location Pattern Matching Diagram

Figure 2.1: Techniques used for Acoustic Sensing

Some works also introduced the concepts of <u>active</u> and <u>passive</u> approaches to touch (or acoustic) sensing, such as TapSense [HSH11].

On active approaches, an explicit object (e.g. a special pen or a speaker) must be used to interact with a surface, which usually requires some electronics to work. Alternatives to this method involve instrumenting the object providing input (fiduciary markers, speakers, wrist-mounted sensors, etc...).

A passive approach allows a much more convenient and simpler interaction with surfaces, as it does not need any active components to work properly. Instead, it relies on the properties of the object used (its structure, material or unique properties).

On the next section we will present some tools that use acoustic sensing to retrieve information about interaction with objects or surfaces.

2.2 Approaches to Acoustic Sensing

Acoustic sensing has been explored as a powerful tool to retrieve information from a user's interaction with a surface or object. Some of this information can be used by developers to enhance interfaces with new features, or most importantly, improve interaction on current interfaces (some of the recurring problems currently observed on interfaces are complex navigation, lack of useful shortcuts, among others).

There are 2 main groups on these approaches: the ones whose main goal is to explore acoustic sensing capabilities for use on touch surfaces, such as touch displays, and approaches that use acoustic sensing to empower simple objects and surfaces, possibly ignoring the touch capabilities they may possess.

2.2.1 Use on Touch Surfaces

This section presents some work done with acoustic sensing with the main purpose of extending interaction on touch surfaces. However, some of these contributions can also be applied to non-touch surfaces or objects.

The most important aspects of each work are presented, as well as some of their advantages, and limitations (when identified).

One of the main objectives for these approaches is to expand user interaction with interfaces, and allow new and powerful interactive capabilities through acoustic sensing.

One of these approaches, and the paper preceding our work is [LJJ11]. The motivation for this contribution comes from the fact that current touch technologies limit the interaction by only focusing on input position and shape. Touching a surface also generates sound, which can be analyzed and used as a method of input.

To achieve this, a sonically-enhanced touch recognition system is proposed. The main principle is that two gestures exercised on a touch display can be identical (even if done with different body parts or objects), but still have different acoustic signatures. In particular, touch produces two main characteristics:

- Attack amplitude, the intensity of impact;

- Spectral quality, the sound timbre.

These two new dimensions can be processed to expand the interaction space, without adding any complexity to user interaction (no additional apparel or complex gestures). To capture the sounds produced by interaction, a contact microphone is used. However, this work also notes that some devices (e.g. tablets or mobile phones) may already have built-in microphones that might be used instead.

The gesture recognition module is implemented in Pure Data, a cross-platform language. The process for gesture recognition is as follows:

1. Sound capture sampled at 44.1kHz;
2. Noise filtering;
3. Signal analysis, which retrieves peak amplitude - **intensity cue**;
4. Calculation of spectral signature - **timbre cue**;
5. Match with database of trained gestures.

This system's performance is similar to optical touch recognition.

The main **limitation** of this approach is that it is more error-prone, because a touch on the device's case or bezel can be interpreted as a user's gesture. This can be avoided by only recognizing highly expressive gestures.

Another similar approach is described on [HSH11], which resulted on a system called TapSense. TapSense is a tool that allows the identification of the object (or body part) used for input on touchscreens.

The **importance** of this work is that interaction with touchscreens (especially on smaller screens) is sometimes hard to carry out, in particular when more advanced options are available (e.g. dropdowns, contextual menus) that might require double-tap or timed taps to the touchscreen.

One of TapSense's **objectives** is that users need not be instrumented to interact with the surfaces, leaning towards a passive acoustic sensing approach. To achieve this, TapSense uses 2 processes: a method to detect the position of the input, and a method to capture, analyze and classify impacts on the interactive surface. It relies on the same basic principle as [LJJ11]: different material produce different acoustic signatures.

Similarly to [HH08], TapSense uses a medical stethoscope and a microphone to capture the vibration spread along the surface when it is touched.

The segmentation process leverages on the fact that stethoscopes provide a high level of noise suppression, and segments the audio data from any background noise by applying a simple amplitude threshold.

The processing is done by applying an FFT to the data (sampled at 96kHz with a sliding window of 4096), producing 2048 bands of frequency power, and all but the lower 500 are discarded (only those with frequencies ranging from 0 to 10kHz approximately are kept). Some other features are also included, such as the average absolute amplitude, total absolute amplitude, standard deviation of the absolute amplitude, and the center of

mass for both the segmented input signal and the FFT. The fundamental frequency of the impact waveform is also calculated.

With the information obtained, TapSense classifies the gesture using a support vector machine (SVM) [Wik14b].

On average, the entire classification process (starting from the time the impact hits the touchscreen) takes 100ms, allowing for **real-time interaction**. However, TapSense has a **limitation** when it comes to almost simultaneous touches on the display. If two input objects hit the surface at a sufficiently close temporal proximity, TapSense will not separate the sound generated for the 2 impacts, and will not be able to classify them correctly.

The ability TapSense has to identify which part of the finger is used for input (presented on 2.2) can be used to avoid 2 significant limitations in touch interaction:

- Advanced options and menus, currently opened recurring to relatively complex gestures (e.g.tap-and-hold period, multi-finger input) can be triggered with a simple tap with a knuckle or nail;
- Functionality currently assigned to new buttons or dropdowns on the interface could be triggered with a different gesture on the interface.

Figure 2.2: TapSense finger input classification

Other type of objects can also be used to interact with TapSense. Using objects allows more flexibility when interacting with the surface, and can be used to various purposes: different actions triggered with different objects, unique identification of a specific object (to assign a set of actions to a single object/single person). The unique identification feature can be particularly useful for collaborative activities, such as document editing.

Harrison and Schwarz, 2 of the main contributors for this work, are both co-founders of Qeexo [Qee14], a company that is commercializing an improved version of TapSense, FingerSense. The main idea is to offer these capabilities at the system level (embedded on smartphones and other touch devices) to reach a wider range of clients. The project has been heavily funded over the years (over $2M as of July 2014 [Sch11]), and the company is currently partnering with device OEMs (Original Equipment Manufacturers) to integrate their solution on a range of devices.

Another tool developed to allow the introduction of acoustic sensing features on interaction is Expressive Touch [PH14]. Expressive Touch is a tabletop interface used to infer and

calculate tapping force using acoustic sensing.

This work adds another dimension to the previous approaches: touch force. The information captured when sound waves created by the user's finger upon impact can also be analyzed to allow variant tapping force recognition while interacting with the surface.

The **motivation** for this work is similar to previously described approaches, relying on the fact that the human touch is expressive and has much more information than that used on touch interfaces, where it is commonly reduced to a simple binary option: touch or no touch.

It also points out that, although some researchers tried to solve this problem using the mobile phone's accelerometers, they still do not have the required resolution to distinguish the granularity of the human tap.

The approach used to gesture recognition is similar to some proposed on other works, such as [LJJ11], but the **goal** is distinct from most of them: on Expressive Touch, the amplitude of the sound produced by finger taps is used, to estimate the associated force. This decision is supported by the fact that humans have fine-grained control over the pressure exerted with their hands [SM59], so this technique can be used successfully.

The setup used for this experiment is similar to the one used on [LJJ11], a touch table augmented with four contact microphones.

The process used to determine tapping force of an input gesture with Expressive Touch is repeated for each of the microphones, and is as follows:

1. Sound capture and sampling at 44.1kHz;
2. Filtering (bandstop filter to eliminate power line hum, and noise filtering);
3. Maximum amplitude (peak) determination;
4. All four peak values are processed and paired with touch events from the table.

The authors conducted a study to determine which technique was best (more accurate) to calculate the peak amplitude value using acoustic sensing. The best approach, and the one implemented on Expressive Touch, is comprised of simply averaging the peak values determined on each microphone.

The main **limitation** of this work is the fact that it may not be easily transferable and used on other tabletop surfaces. The setup's structural properties may compromise the sound capture by the microphones.

A more complete approach was developed on [McC+12], adding acoustic sensing to an existing interactive system, MIST. MIST is a multi-input sensing table, that leverages on the additional information that is provided by multiple sensors to collect data relevant to interaction with the table's surface.

Most touch tables rely on a single camera for input location and interaction. MIST adds acoustic (piezoelectric) and pressure sensors to this setup. Adding to a camera with an infrared filter, the table is equipped with four pressure sensors and four acoustic sensors mounted at the corners of the table.

All data generated by the sensors are interpreted and processed by computer. Data generated by the camera is processed with Community Core Vision (CCV), while data generated by the pressure and acoustic sensors is parsed with Pure Data [Puc+96], and sent through UDP packets to the local host.

Acoustic sensors

The authors analyzed how well these sensors locate touch on the table to optimize the interaction.

Retrieving information from all acoustic sensors could be useful to triangulate a touch, but would introduce extra latency on the system in order to wait for the touch to be detected by all sensors. Instead, a different **approach** is used on this system: the table is divided into quadrants, each one with a sensor (that is responsible for detecting touches on that quadrant). To avoid sensing a signal twice for the same tap (detected by another sensor on another quadrant), sensors must stop sending information until all of them detected the wave. Considering the dimensions of the table, and the sound propagation velocity on the material (acrylic), a timeout can be calculated (0.39 ms, with the setup used on this case).

The **problem** with this approach, although simple to implement and relatively efficient, is the loss of performance on the dividing zones between quadrants. When struck on this zones, results are more ambiguous. To avoid this, sensor data is tuned with Pure Data, which optimizes the definition of the four quadrants.

Pressure sensors

Four pressure sensors are mounted underneath the surface. The pressure applied on the surface deforms the sensor's material depending on the pressure applied. The sensor then returns a value for the pressure exerted. The time it takes for the sensors' material to return to its original shape does not allow for an efficient detection of quick taps on the surface, making the acoustic sensors a better alternative.

The drumming application implemented to test MIST uses all sensors, with different combinations separated into modules. Acoustic sensors are mainly used (depending on the module) to locate gestures and trigger sounds. A division of the surface into two or four quadrants was tested, with two quadrants division being the one who yields the best results.

2.2.2 Use on Non-Touch Surfaces or Objects

This sections presents systems that explore acoustic sensing on surfaces and objects, independently of the touch capabilities they might possess. This possible loss of capability allows for other type of research to be done, in particular it allows the empowerment of simple objects and surfaces, such as tables or walls to become interactive.

Although our approach does not focus on non-touch surfaces, some of these works have introduced very interesting techniques and methodologies that can be leveraged on our approach.

One of the main works on this area that first proposed the use of acoustic sensing to expand interaction with simple objects or even walls and windows, is ScratchInput [HH08].

ScratchInput is an acoustic-based method that relies on the unique sound that is produced when a fingernail or other object is dragged over a surface, such as a table or a wall.

This approach is based on the fact that every gesture requires some movement with a finger or other object, and that these movements interact with the surface to produce a particular sound. Each one of these sounds has an unique acoustic signature that can be captured and analyzed.

This analysis returns some important properties of the sound: amplitude and frequency. The recognizer then uses a shallow decision tree to decide on which gesture was made, based on the signal peak count and amplitude variation.

The main **advantage** of this tool is that it only needs a simple and inexpensive sensor attached to the device's microphone.

Its main **limitation** is that, as it only relies on sound input to distinguish the gestures, some movements that have different meaning (e.g. writing similar letters or numbers) cannot be distinguished, as they sound the same. Nevertheless, there are still dozens of gestures that can be successfully recognized.

To prove this concept, some example applications were developed, and the tool tested on different surfaces. The authors concluded that interaction carried on surfaces made of glass or with glossy surfaces was not very effective (no friction or not enough to produce a recognizable sound).

Finally, the authors did an **evaluation** of this tool using 15 users. The average accuracy on gesture recognition was 89.5%, and it was concluded that accuracy suffered as as gesture complexity grew.

Another similar system was proposed by Paradiso et al. [Par+02].

However, the main **objective** of this work is to track knocks and taps atop large glass surfaces, which was not possible on [HH08]. To achieve this, acoustic sensing techniques are used, recurring to four contact piezoelectric pickups, used to record the impact produced by touch impacts on the surface. Acoustic sensing was used on this work due to some of its **advantages**:

- Scales relatively well to large-scale surfaces (more pickups simply installed if needed);
- No noticeable decrease in performance with surfaces' deterioration (scratches or cracks).

The signal retrieved by the described setup is processed and the following characteristics are retrieved: amplitude, frequency components, and differential timings. With this information it is possible to estimate location and position of the hit, its nature (what gesture was made, e.g. knuckle tap) and its intensity.

The main **motivation** to this work is to enable existing surfaces (large sheets of glass, such as windows) to become interactive. This is a considerable **contribution**, allowing such interaction on glass surfaces, as wave propagation characteristics are more complex (dispersive propagation) than on other surfaces, because of the nature of the material. Due to this added complexity, approaches used for acoustic sensing on other surfaces are not successfully applied to glass surfaces, and so, signals for glass need to be digitized and processed **algorithmically**.

Using MATLAB, the authors obtained much more robust performance. This boost in performance was achieved by cross-correlating the signals across pairs of sensors and extracting the differential time from the correlation peak. However, due to the signal's distortion while propagating through the glass, results of the correlation might be ambiguous. This problem was solved through the redundancy added to the system (additional sensor).

To distinguish between knuckle taps and "metal" taps (hard objects), the signal's frequency is estimated by counting the number of times the sensor's waveforms cross zero during a fixed interval. As Figure 2.3 illustrates, the frequency distributions are quite separate, which makes this classification rather simple. For additional reliability, the differential timing between sensor pairs is also taken into account.

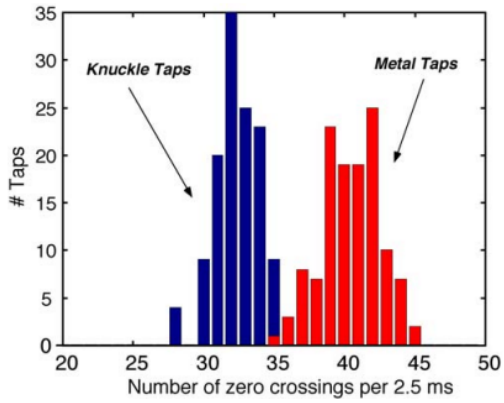


Figure 2.3: Histogram of knuckle and metal tap frequency

However, this approach has a clear **limitation**: more similar gestures (like a knuckle tap and a finger tap) may not be distinguishable with this analysis.

The previous approaches rely on passive acoustic sensing to carry on interaction with the surface. As said before, this allows for more flexibility and freedom while interacting with the surface, as no extra apparel is needed and the user can use the surface in a simple manner. However, these approaches are more constrained by properties of the surface or the information it is possible to obtain from the interaction.

Other approaches use devices and other objects to explore acoustic sensing, leaning towards the use of active acoustic sensing techniques.

One of these cases is Stane [Mur+08], a hand-held interaction device controlled by rubbing or scratching touches. The main **objective** is to classify these gestures in order to make them usable for interaction with surfaces.

The authors consider that most capacitive sensing technologies (touch displays in particular) do not give reliable and sufficient feedback to users. This makes interaction without visual contact rather difficult.

Stane makes use of a contact microphone attached to the inside of the device to pick up vibrations generated by touching the device case. The device interior is formed by different textures so that each texture can be easily distinguished when touched (different vibrations picked up by the microphone) and allow more interaction options. Also, this gives user

different feedback for each different texture. The Stane device is presented on Figure 2.4.

Figure 2.4: Stane device

The contact microphone is attached to the inner exterior of the device, and the information retrieved is passed to a SHAKE (Sensing Hardware Accessory for Kinesthetic Expression) module that includes a high-impedance microphone data acquisition circuit and a vibration driver. The microphone bandwidth was limited to 2kHz as there is little information above this frequency. After the information is passed to the SHAKE module, the SHAKE micro-controller filters, samples and packages it to be sent to the host device (through the Bluetooth port).

The **classification process** occurs in two stages. Before classification, input audio is windowed with a Hamming window, 512 samples long, and the Fourier transform is calculated.

The first stage, a low-level instantaneous classification process, takes the features output by the last action, and matches it against a set of four different trained classes (based on the various textures on different regions of the device). This first process identified the different regions of the device with a 75% accuracy.

The second stage is a dynamic system, used to smooth out the fluctuations on the classifier. This stage was not further developed on this paper, but a complex recurrent classifier already available could be used to achieve the effect desired.

The work done on Touch & Activate [OST13] also uses active acoustic sensing.

The main **objective** of this approach is to allow the simple and efficient prototyping of interactive objects that have input touch capability. This provides developers with all the needed hardware configuration to the development of gesture-based systems.

This is as an active acoustic sensing approach to the gesture recognition problem because two active components are attached to the object: a speaker (to produce vibration signals) and a microphone (to capture vibration response), paired as a sensor. This kind of approach is sometimes discarded due to the amount of instrumentation they might need. However, the components Touch & Activate uses are very simple and small.

This system is based on the fact already explored by some of these approaches: each object has individual resonant properties, based on its properties (e.g. material and shape). When an object is touched, the surface is affected, and the way vibrations resonate through the object is also affected. By producing vibrations on the object and capturing them afterwards, such changes can be captured and analyzed.

The software consists of three modules: Sweep Signal Generator, FFT Analyzer, and

SVM Classifier.

The Sweep Signal Generator is responsible for generating sinusoid sweep signals through the vibration speaker. This vibrations are spread throughout the object and used on the process of detecting touch on the object's surface. The sound produced, on the range from 20kHz to 40kHz, is inaudible to humans.

The FFT analyzer is responsible for processing the vibrations captured by the microphone into the resonant frequency response. This module samples audio at 96kHz, the same rate generated by the Sweep Signal Generator in order to obtain the signal produced that ranges in 20kHz-40kHz. Then, 400-point relevant features are extracted representing the range indicated with down-sampling. Non-relevant values (with values nearing 0) are discarded, to speed up the recognition process.

The SVM classifier is responsible for the classification process. It uses a Support Vector Machine (SVM) [Wik14b] to process all data that is generated by the FFT analyzer. This data is passed to the classifier every 20 ms. Before the classification, the classifier must be trained for each gesture. Touch & Activate provides developers with a training mode to fasten the training process.

An interesting **application** of Touch & Activate is the possibility to use this system on mobile devices, to improve the way we interact with touch displays. This allows hand posture recognition while we handle the mobile device, and pressure recognition while we interact with the display. Posture recognition is useful because the hand posture employed while using the mobile device can differ considerably depending on what functionality or application is being used (e.g. the postures used to make a call or to take a picture with the camera are clearly different). This fact can improve usability by, for example, launching an application automatically when the corresponding posture is recognized. Pressure recognition can also improve interaction. Although single-handed interaction (especially with the thumb) is still preferred by users [KBC06; PKB06], and therefore most applications use single-touch operations to meet up with this preference, this evidence limits interaction with mobile devices. By recognizing different levels of pressure, developers can improve interaction by assigning tasks to each recognizable pressure level (e.g. pressure-based zoom).

Most of this system's **limitations** are related with the properties of the object it is applied to. The materials that compose the material can conduct the vibration differently. Elastic materials such as rubber damp the vibrations, which affects the capture of vibrations by the microphone. The size of the object can also hinder the recognition process. On bigger objects, the sensitive area can be small (limited to the distance between the speaker and the microphone) and as such the system may not work properly without additional sensors spread through the object's surface. If the object is small, the instrumentation required might be obtrusive to the interaction with the object. The shape of the object, in particular its symmetry can hinder the recognition process. This happens because symmetric touch gestures on a symmetrical object are not distinguishable, as they have the same resonant frequency if the sensor is placed on the axis of symmetry. The solution is to place the sensor away from this axis.

Another more recent work that explores acoustic sensing is SurfaceLink [Goe+14], a

system that allows users to, using natural surface gestures, control association and share information amongst a set of devices.

For this, SurfaceLink uses on-device accelerometers, vibration motors, speakers and microphones. This work proposes that we can leverage on these components to achieve inertial and acoustic sensing capabilities, that enable multi-device interaction on a shared surface.

On Figure 2.5 an example of interaction is presented.

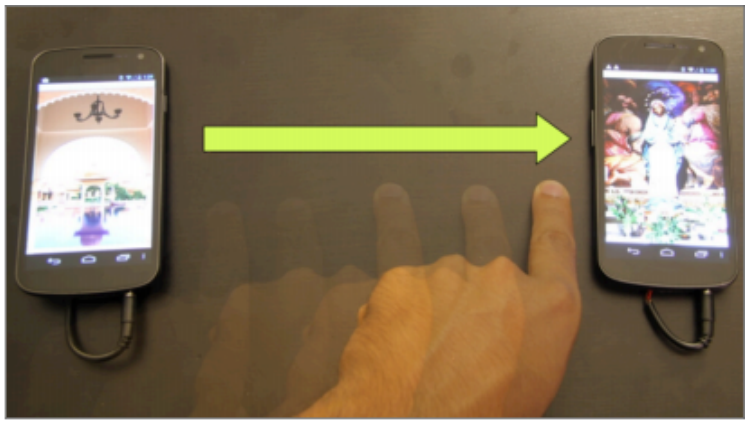


Figure 2.5: SurfaceLink user interaction with two mobile devices

To what **acoustic sensing** is concerned, SurfaceLink uses the same principle already introduced on most works previously described: a finger or hand dragged across most surfaces produce vibrations. Those vibrations can be captured by a built-in microphone or, for additional sensitivity, a contact microphone. These vibrations have different characteristics, e.g. they are louder when near the device and softer when far.

By further analysis, one can conclude that a gesture direction can be retrieved from the spectral analysis. In particular, when a user's finger comes near the device, the resonant frequency decreases, and the opposite occurs when it moves away. This can be used to distinguish continuous gestures (and different gestures' intentions).

They also discovered that the speed of a gesture on a surface is directly correlated to the amplitude of the resonant frequency. When the user does a fast swipe, the gesture accelerates and decelerates quickly. The slope of increase and decrease in amplitude is also steeper. This allows for a differentiation of gestures' speeds.

Another finding, coupled to the speed of a gesture, is the gesture's length. Considering that SurfaceLink is used mostly for the interaction between devices, the length of the gesture is bounded by the distance between the devices. It is then possible to compare gestures with different lengths based on their duration.

Other contribution of this work is the fact that SurfaceLink is able to distinguish the shape of a gesture. By analyzing the spectral information and performing pattern matching, it can infer a fixed set of gesture shapes, such as lines or squares.

The SurfaceLink **process** for detecting surface gestures is as follows:

1. Each device records audio at 44.1kHz;

2. Data segmented by thresholding between 5kHz and 15kHz;
3. Generation of machine learning features;
4. Gesture classification through machine learning (using retrieved features);
5. Decision based on the classification result of each device.

The **machine learning features** are generated by producing the magnitude spectrogram of the audio signal, and using a fast Fourier transform (FFT), with a 100 sample Hamming window [Dac03]. Next, the $|FFT|$ data is down-sampled into 6 frequency bins and 10 time frames, to reduce number of features, yielding 60 gesture features (6x10 matrix). The temporal data is then put through a band-pass filter (5 kHz to 15kHz) and down-sampled into 10 time windows. This captures the temporal variation for each gesture, and provides 10 additional features. Finally, 2 more features are retrieved: the total energy in the first and second half of the gesture.

These features are used to inform a kNN classifier [Wik14a] (with k=2) on each device. Separate kNN classifiers are used for each gesture property (class, length, touch mode, and shape), to reduce the number of training samples.

2.2.3 Issues on Touch and Non-Touch Approaches

Most of the studied approaches present the introduction of acoustic sensing and gesture classification capabilities as a way to improve interaction on surfaces, either by adding acoustic sensing features to other available technologies (for example touch displays or pressure sensors), or by empowering simple surfaces (such as walls and everyday objects) with acoustic sensing to achieve a multitude of interaction possibilities with these surfaces.

However, there are some common issues to using acoustic sensing.

When more complex gestures are considered (drawing letters or numbers for example), successful classification is quite limited. None of these approaches provides a solution to this problem, mostly due to insufficient information retrieved from acoustic sensing. However, this issue can be overlooked, as interaction is often best carried using simpler touches (e.g. a tap or a knock).

The nature of the surfaces can also be a problem. For touch approaches, touch displays are relatively durable surfaces, so this problem is also solved. For non-touch approaches, some surfaces (such as wood tables or walls) can get scratched or deteriorated, which compromises a successful classification.

Considering these facts, it is important to study the devices that provide these surfaces, and analyze their characteristics and capabilities, specially focusing on the type of surface and technologies used for touch capabilities, and how acoustic sensing can be processed on these surfaces (in particular how audio processing can be executed on these devices).

2.3 Devices: Surfaces and Capabilities

As the approach developed will focus on two main types of devices, mobile handheld devices, such as tablets or smartphones, and tabletop touch devices, with wide surfaces, it is important to understand the characteristics of each of these types and the surfaces used for interaction.

In this section we describe what are the main characteristics for each of these types.

2.3.1 Mobile Devices

A mobile device is a small computing device that can be used handheld. It is usually equipped with a touch-sensitive display.

Surface

A mobile device's surface is usually small (to allow handheld use), and its screen occupies almost all of the front surface.

A mobile device's touchscreen is usually built with glass. This decision comes from the fact that glass does not scratch as easily as plastic, it feels more comfortable for tactile interaction, and it is overall more resistant. This hardware detail also explains why the previous studied approaches work on a wide multitude of these devices: sound waves propagate very well on glass surfaces.

The touchscreen technologies used on these devices can vary, however there are 2 main types: resistive and capacitive displays. The main difference between these 2 types of touchscreen technologies is the way they process interaction with the surface: resistive touchscreens rely on pressure exerted on the surface, and capacitive touchscreens make use of the electrical properties of the human body. This difference makes capacitive screens more sensitive to touch, but prevents users from using the device with any non-conductive object (such as a pen or while wearing gloves).

Audio processing

Processing audio to develop acoustic sensing features on mobile devices can be considered a more complex task than on touch tabletops or fixed displays. This process starts with the need to capture the audio.

One of the main problems that arise from this need is that mobile devices were designed to be ubiquitous [Bal+06], and so they can be used under extremely diverse conditions. Mobile devices' microphones do not work well under noisy or chaotic environments, and so it is not recommended to use them to record surrounding environment audio. According to [Hec14], audio filtering algorithms used on these devices can be sufficient to remove background noise, but sharper, irregular disruptions can still hinder sound quality.

Another problem also pointed in [Hec14] is that the quality of the sound recorded with a mobile device microphone is not as reliable as a dedicated sound capture device. This is in

part a design problem, with technological companies nowadays more oriented towards improving the design or other functionalities of the device, not focusing on improving essential features such as sound capture (specially on smartphones).

Although the quality of the sound recorded is not ideal, it may still be sufficient to allow the gathering of important information. However, the input signal still needs to be filtered to increase the quality of the results. This filtering process must be done with caution, so as not to leave out any relevant piece of data.

2.3.2 Interactive Tabletops

Interactive tabletops are tables or surfaces augmented with technologies to allow interaction with a installed system or interface. These devices are usually encountered at museums, shopping centers, and other public areas, and are used as space exploration virtual guides or context information providers.

Surface

Tabletop surfaces are usually wide and designed to be interacted with the whole hand or even both hands. On Figure 2.6 an example of one of these surfaces is presented.

These surfaces can be quite similar to the ones found on tablet devices, however it is not uncommon to encounter tabletop surfaces composed of a computer and a touch overlay that captures the touch events. These touch overlays reproduce the touch capabilities of common touchscreens, but they also increase the amount of configuration needed to obtain a functioning touch device (additional drivers and software).

The larger dimensions of these surfaces allow for a more diverse level of interaction, with the possibility of using more complex gestures, and wider parts of the body.

Similarly to mobile devices' surfaces, these surfaces are usually built with glass, which propagates the sound very well, and reduces the need for additional sound-capturing devices if they are used on wider surfaces.

Figure 2.6: Example of a tabletop touch surface

Audio processing

As most of these devices do not include a sound capturing device, an external microphone is required.

This can be beneficial to the development of gesture classification and acoustic sensing approaches, as external microphones are usually more reliable than built-in microphones (and can be replaced with ease). However, this creates another concern: the location of the microphone on the surface can influence the results of audio capture and consequently the gesture classification resuls.

The added reliability of external microphones significantly reduces the need to filter the captured sound, which simplifies the development for this kind of surfaces.

2.4 Toolkit Development

This sections presents toolkits that allow the incorporation of gestures on applications. Although the approach used is not the same as our contribution (they are not based on acoustic sensing capabilities), some components are similar. In particular, the approaches considered on each work, techniques such as algorithms and architecture designs and also limitations of each toolkit provide greatly valuable information as input to our research.

GART

GART [Lyo+07] is a toolkit that allows the development of gesture-based applications.

The main **objective** is to provide a tool to application developers that can encapsulate most of the gesture recognition activities (data collection and the use of machine learning algorithms in particular). The system's architecture is composed of 3 main components: Sensors, that collects data from sensors and can also provide post-processing of the data;

the Library that stores the data and provides a way to share this data; and the Machine Learning component, which encapsulates all training and classification algorithms.

Sensors are the components that interface with the hardware, collect data and may provide parsing or some post-processing of the data. GART sensors support sending data to listeners in two formats: samples and plain data. A sample is a set of data that represents a gesture, and it can also contain some meta-information such as user name, timestamps and notes. GART also allows the integration of new sensors, by inheriting from the sensor class. This toolkit provides three built-in sensors: mouse, camera and accelerometers. For the camera sensor for example, the sensor provides post-processing that tracks an object based on a color histogram. Samples produced correspond to the object (x,y) coordinates on the image.

The Library component is responsible for storing and organizing data. It acts as the database for the applications. The data used for the training process is stored here (labeled gestures). This component also allows exporting of samples to XML.

The Machine Learning component encapsulates all processing done with machine learning algorithms. It is used both on the training (data modeling) and gesture classification processes.

This component receives the samples from the sensor and returns the result to all listeners (the application). This result can be the classified gesture, or an error that might have occurred. This system's gesture classification module uses hidden Markov models as default, but it allows for expansion with other techniques.

In the end, with a relatively low number of lines of code, an application can use GART to recognize gestures. Still, there are some considerations that the developer has to take into account when developing an application that uses GART. For example, the developer must choose an appropriate (and make available) sensor to collect the data that is needed to the application.

iGesture

iGesture [SKN07] is a Java-based gesture recognition framework, whose main goal is to help application developers and recognition algorithms' designers on the development process.

The main **advantages** of this framework are:

- iGesture provides a simple application programming interface (API) to an easy access to all its features. This hides all implementation details (especially the recognition process, that can be quite complex) from the developer;
- It is extensible. Although it was developed using digital pen and paper technology, all the framework works on an abstract input device interface.

The framework is composed by 3 main components: the Management Console, the Recognizer, and Evaluation Tools. All these components require an additional one with all common data structures.

To allow flexible and simple use and design of algorithms, an `Algorithm` interface is provided, with methods for initialisation and the recognition process itself.

An algorithm is initialised with an instance of the `Configuration` class, which contains gesture sets, the collection of algorithms and their specific parameters. This object can be created with the Java API or by importing data from a XML file. These features are more relevant for algorithm designers.

For application developers, the key component of iGesture is the `Recogniser` class. This class encapsulates all functionality provided on the Recognizer component, and is initialized with a configuration object (that provides information about algorithms to be used) loaded from an XML file. This class allows the use of multiple algorithms (if more than one was specified on the configuration object). Depending on the method called, the recognition process can use the result of the first algorithm that returned valid data, or combine the results of all algorithms.

The data structures used for representing gestures within the iGesture framework allow the representation of single gestures and groups of gestures. Furthermore, as different algorithms can be used, this representation must be general and flexible. The `GestureClass` class represents the gesture, characterised by a name and a list of descriptors. These descriptors can be textual, images, or samples. Textual descriptions specify the directions between points of a gesture, image descriptors are not used on the recognition process, but serve as a visualization of the gesture (to use on user interfaces for example), and samples are used by training-based algorithms. Gestures are then grouped in a `GestureSet`, used to initialise an algorithm.

The storage mechanism (to allow later retrieval of gesture samples) is also implemented in a way that all interaction with the data source is done through a well-defined interface. The primary storage engine used was db4objects (now discontinued). iGesture also allows the serialization of all data into a XML file, using x-stream Java library [Cod].

iGesture also includes a management console, a Java Swing application that can be used to test gestures, create new ones, and create test sets. This component can acquire the gesture captured from the input device, and includes the following main features:

- Execute the recognizer with the gesture set and selected algorithm to test the acquired gestures;
- Add gestures as samples of a given gesture class;
- Create, edit and delete gesture classes and manipulate descriptors;
- Create test sets to evaluate and compare algorithms, and export it to an XML file.

A screenshot of this console is presented on Figure 2.7.

Finally, it is also possible to generate gestures interactively based on a set of defined gestures, by using the digital pen, to simplify the process of capturing gesture samples.

Google Gesture Toolkit

Google developed a Gesture Toolkit [Li09] that simplifies the creation of mobile gesture applications.

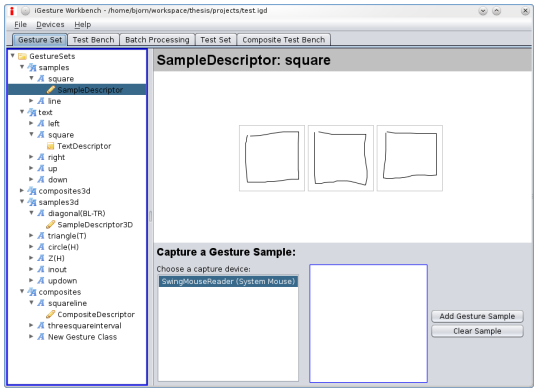


Figure 2.7: iGesture Management Console

The **motivation** to this work comes from the fact that human-computer interaction based on the WIMP (window, icon, menu, pointer) paradigm, although powerful and widely used, does not allow for eyes-free interaction, and can be slow to use on certain applications. This is aggravated where mobile phones are concerned, which have relatively small displays, and are used on the go.

Authors consider that the current expressive power of gesture-based interaction is not great, as most mobile phones support only a small set of gestures. This can be further improved by allowing developers and end users to create their own set of gestures.

Thus, the two main **challenges** this toolkit addresses are:

1. Gesture-based interaction requires a different mechanism for event dispatching and processing than WIMP-dominated mobile interfaces;

2. The gesture classification process is relatively complex, involving machine learning techniques, which can be difficult to use, or time-consuming.

The first challenge is solved by encapsulating low-level details of gesture composition and rendering. The second challenge is addressed by packaging all these features and offering them as a simple and well-defined interface.

A gesture overlay (a transparent layer stacked on top of the interface widget) is used to collect touch events, wrap sequences of movements into gestures and sends them to the application. It also moderates event dispatching of the entire interface, by disambiguating gesture input from regular touch input.

The toolkit provides two different recognizers: one that recognizes the English alphabet and another that recognizers developer or user-defined gestures (customizable recognizer).

The English alphabet recognizer collects data using GestureSampler, a data collection tool. Google collected data from a set of users (to capture the variation of user's writing style), and used it to train a neural network classifier to recognize the letters. This recognizer only recognizes a fixed set of gestures, but does not require further training.

The customizable recognizer must be able to learn and acquire gestures both at development time and also at runtime (end users gestures). To achieve this, the recognizer uses

a nearest-neighbor approach to match an unknown gesture against a database of known gesture samples.

The extension of a set of application-specific gestures at development time is enabled though a gesture library, that omits all gesture classification details.

Gillian & Paradiso Gesture Recognition Toolkit

Gillian and Paradiso [GP14] developed an open-source C++ library that enables simple and quick gesture recognition.

The **motivation** for this work resides on the fact that gesture recognition is becoming an important and powerful tool for human-computer interaction. Most smartphones and tablets make use of touch and gesture interaction to allow a more efficient and interactive navigation on user interfaces. This leads to a wide interest on the development of gesture-based applications by not only professional developers, but other individuals.

The **problem** on which most of these individuals, and even accomplished engineers and developers, stumble upon is the difficulty on implementing and understanding gesture recognition techniques, and also achieving real-time performance. Even when this part is accomplished, all the supporting infrastructure needed for building the application (such as data preprocessing and relevant features extraction) can become a problem on the development process.

To address this problems, the authors created the Gesture Recognition Toolkit (GRT). The core design principles for this toolkit are:

- **Accessibility:** Functionality is obtained without the need for many lines of code, as GRT provides a consistent and minimalist design. This allows an easy use of the core functionality;
- **Flexibility:** An object-oriented architecture allows an easy separation of concerns into core modules, including a gesture-recognition pipeline. Algorithms can be used as stand-alone classes. Some of the modules included are: preprocessing, classification and post processing.
- **Choice:** The developer can choose which algorithm is best to recognize its specific gestures. Also, this switching is done on a seamless way, with minimal modification's to the code;
- **Supporting Infrastructure:** The preprocessing and mining of relevant data can be a complex and time-consuming task, even if the recognition process is taken care of. To avoid this, GRT provides tools for preprocessing, feature extraction and selection;
- **Customizability:** Leveraging on its object-oriented architecture, the toolkit allows the integration of the developer's own algorithms, by simply inheriting from one of the base classes;
- **Real-time Support:** One challenge is to achieve real-time gesture recognition. Machine learning algorithms (that are previously trained data) can automatically recognize valid gestures from a continuous stream of real-time data.

With all this design principles assured and spread throughout the toolkit, this contribution makes gesture-based applications much easier to develop and powerful.

2.4.1 Discussion on Toolkits

After studying all of these toolkits, there are some conclusions to be taken.

All of these contributions come from the understanding that human-computer interaction has changed on the past few years. Interaction can now be carried on other more creative and useful ways. One of these ways is by using more complex gestures to interact with an interface (e.g. a tablet touch display). To achieve this, these contributions propose gesture classification toolkits to improve the development of gesture-based applications.

Some of the characteristics can be compared among all the toolkits. This comparison is presented on Table 2.1.

Characteristics	**Toolkit**			
	GART	**iGesture**	**Google Gesture Toolkit**	**Gillian & Paradiso**
Export and reuse of gestures	X	X	X	X
Support for complex gestures (e.g. letters drawing)			X	
Extension of features	X	X		X
Component modularity	X	X	X	X
Ease of Integration (readable code, amount of added coding)		X	X	

Table 2.1: Toolkit Comparison

This table showcases the main similarities and differences between the toolkits. Component modularity (at an architecture level) and the need to export and reuse the recorded gestures are the main points of contact, and a verified need for all these toolkits. Where complex gestures support is related, only Google recognizes more complex gestures, such as letters and complex shapes, while providing simple integration methods.

Some other important and shared concerns are: the need to provide a well-defined interface for the toolkit to be used in a simple manner and the importance of hiding details from the classification process (machine learning techniques and other algorithms).

In particular, the last analyzed contribution provided by Paradiso et Al. [GP14] describes some important design principles that were used while developing their toolkit. To our work, we consider that the ones that must be mainly taken into account are: accessibility and ease of use, flexibility and separation of concerns, and real-time support.

However, none of these toolkits use acoustic sensing capabilities, and cannot exploit some of the possibilities it can provide to further improve interaction (for example the estimation of gesture force or continuous gestures). Furthermore, there are no available fully-developed toolkits that use acoustic sensing capabilities.

This lack of alternatives for acoustic sensing toolkits makes it difficult for application developers to use acoustic sensing on their applications. On the next chapter, we will describe our approach to this problem, taking into account the characteristics and problems of similar toolkits, and leveraging on the research done.

Chapter 3

Approach

As we have seen in the previous chapter, there are no available toolkits that provide acoustic sensing capabilities that can be leveraged by application developers. However, there are some alternatives for gesture classification toolkits.

The developed approach can be inspired by some of these alternatives, as the characteristics they are built on are common and applicable to a wide range of toolkits (mainly where software requirements and architectural decisions are concerned).

In this chapter we will explain the proposed approach to this problem.

First a brief rationale is presented, highlighting the main ideas behind the approach, how they are going to be executed, and the main decisions taken at the beginning of the development process.

3.1 Rationale

The main idea is to develop a robust yet simple system to capture, analyze and classify the gesture applied using the sound produced by interaction with a surface.

As already stated, this approach will focus on 2 main types of devices: devices with wide touch surfaces, mainly tabletop touch surfaces, and devices with smaller touch surfaces, in particular mobile devices such as tablets. This decision can be justified by understanding some characteristics about each of the considered device types:

Wider touch surfaces, such as tabletop surfaces, allow a much more dynamic and effective interaction with the system or applications being used. On the other hand, smaller touch surfaces are more available to everyone, and the need for better interaction techniques is greater, considering how reduced the size of the touch surface can be.

These 2 main types encapsulate most of the devices with touch capabilities available nowadays.

In the next section the architecture of the approach will be described.

3.2 Architecture

Taking all considerations identified on the previous sections, the architecture of our approach can be projected and developed.

On this section we will focus on the architectural decisions for the approach, and the design principles it follows. We will describe the main processes that compose our toolkit's architecture and some important implementation details.

Some additional overview will be provided to the tabletop version of our approach.

3.2.1 Overview

Before further design and development, it is important to define what are the main software requirements that must be enforced and conveyed by our architecture.

Some of the requirements may be shared with other development toolkits such as ours, and others will be more specific to our approach.

The main requirements are:

- Modularity

  A development toolkit must allow for a relatively high degree of customization and extension with small effort from the developer. In particular, our toolkit must separate each of its components based on functionality, and enforce a simple and obvious use of these components;

- Ease of Integration

  For the toolkit to be accepted and used among application developers, the integration process must not constitute a troublesome task while developing an application. To achieve this, both the architecture and the implementation must be designed to expose only the needed components (classes, methods or attributes) to the application developer;

- Real-time support

  This is a more specific requirement. Our approach focuses on associating sound capture and processing with user interaction, so all of the architecture components must support real-time processing and feedback, to a risk of not delivering a smooth interaction.

Taking these first requirements into account, the architecture can be designed and developed. The approach's architecture (featuring the most important components) is presented on Figure 3.1.

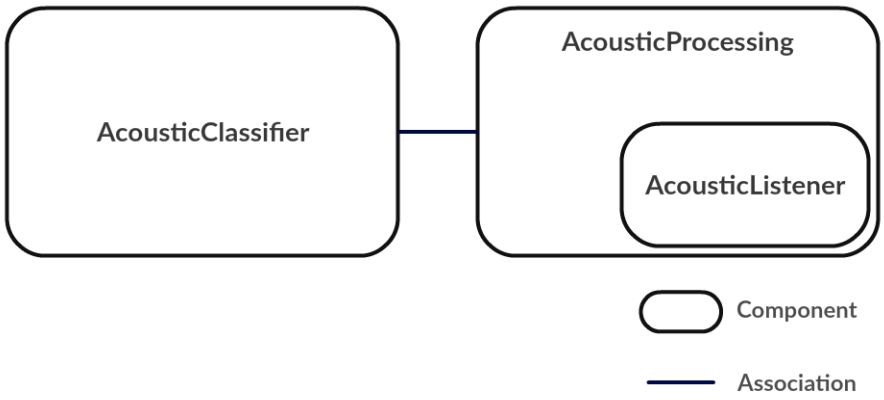


Figure 3.1: Architecture overview

Technologies

To develop this approach, it is important to understand what are the technologies available to develop this toolkit and enforce the requirements identified, and also how can they work together.

This toolkit will be developed for 2 types of devices: small touch devices and large touch surfaces. The architecture supporting the implementation is the same, differing only on the implementation details. The toolkit is developed on Java, an object-oriented language. This allows for a much greater reuse of code while developing for small touch devices (devices that use the Android OS, in particular), and larger touch surfaces (focusing on the Windows OS).

In terms of audio signal processing, the decision was to use **Pure Data** [Puc+96], because is a simple but powerful visual language that allows an acceptable degree of signal processing while providing a simple interface, and it has already been widely used on some other works, as described on chapter 2.

There is also a need for a tool that provides machine learning features, especially supervised learning and classification capabilities. Here, Pure Data is again at an advantage against similar tools, as it provides an option to this need: the **[bonk~]** object. This object includes beat detection and classification algorithms, exactly what is needed for the approach. Also, no additional software or configurations are needed. A brief description of Bonk's technical details and capabilities is provided on section 3.3.

To combine the use of Pure Data with the approach, we use libpd [Bri+11], a library that allows embedding of Pure Data patches in a simple manner. As it is currently available for Java development, it is a perfect fit for the toolkit development (focused on the Android operating system).

Having all these factors incorporated on the design of the architecture, we can now present the architecture and describe its components.

The architecture was designed to be modular and avoid unnecessary dependencies between its components, enforcing the software requirements previously defined.

The **AcousticProcessing** component encapsulates all the signal processing workflow. It is responsible for starting the audio processing service (as a background service) and its audio input channels, and capturing the audio. It also provides a simple API to communicate with the developed audio processing patch and receive or send information to the running service.

The **AcousticListener** component is responsible for implementing the receiving and handling of messages from Pure Data. It is a simple event listener, that is activated when new messages are received. These messages are sent from the Pure Data patch and processed as needed on the toolkit's implementation.

The **AcousticClassifier** is the primary component of this approach. It uses all other components to collect and process information obtained from the devices' microphone, and sends information to AcousticProcessing component to be delivered to the Pure Data service. This component also implements all the methods provided to integrate the developed toolkit on other applications (described with further detail on subsection 3.2.6.

These components encapsulate all the processes needed for this approach. These processes are: data collection, audio processing and filtering, gesture classification, training process and finally, the integration process. These processes follow a sequential flow, as all are essential for the approach to work. An exception is the training process that, although essential to this approach, can be overlooked by simply leveraging on already trained gestures. This flow is presented on Figure 3.2.

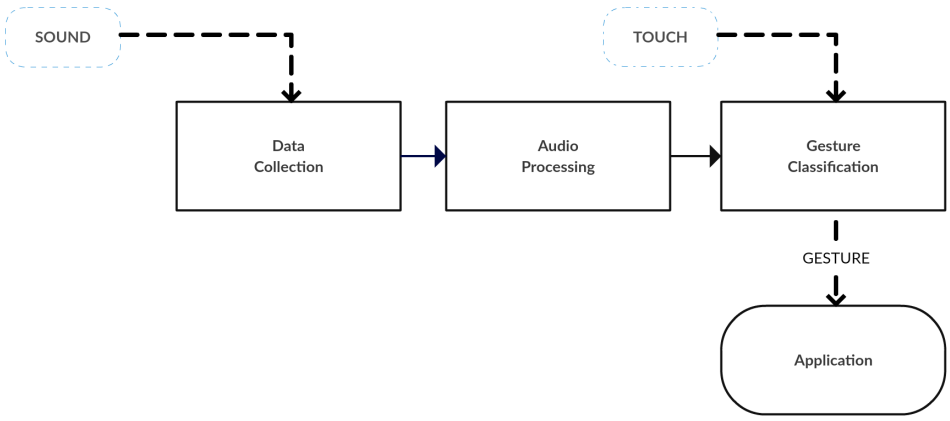


Figure 3.2: Approach's process flowchart

This is how the approach is designed. This processes are now presented in order, with the details and concepts behind described.

3.2.2 Data Collection

The first process of this approach is data collection.

There is some information that needs to be collected:

- The audio signal produced by the interaction with the surface, to be processed, and classified by Bonk;
- The touch event information, mainly its location on the surface, provided by the touch device.

The audio signal is acquired from the device's sound capturing device (e.g. a microphone or a pickup). The AcousticProcessing component receives the audio and passes it to the developed patch (see Figure 3.3 and Appendix A for more detail), who then processes the audio.

The touch event information is collected from Android touch event listener. This listener detects touch events and allows the acquisition of information related to a specific touch on the surface. In particular, we are interested on the touch's coordinates on the screen.

To each of these pieces of information, a timestamp attribute is added, to guarantee synchronization between both sound and touch events when the gesture detection algorithm is applied. This is the information needed for the gesture classification to work.

Some specific implementation-related information is also needed, in particular, the current state of the application using the toolkit. This context (encapsulated on the Context object on Android) provides information about the current state of the application, and is needed for the AcousticClassifier and AcousticProcessing components to initialize the Pure Data service.

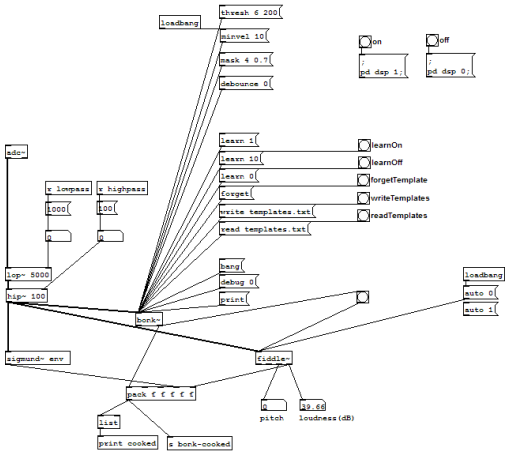


Figure 3.3: Pure Data patch

3.2.3 Audio Processing

After capturing the audio signal, the toolkit must process it to obtain only the information required for the gesture classification feature. As explained on section 2.3, a mobile device can be used almost anywhere, so the captured audio can be composed of mostly noise.

To prevent this issue from harming the classification process, it is important to understand what kind of gestures are going to be recognized and classified. As seen before, most toolkits dedicated to gesture classification focus on simple gestures, or basic shapes,

avoiding overly expressive or complex gestures. This approach is going to follow this decision, because of the already known difficulty to obtain reliable results from low quality audio signals.

To decide on which gestures can be used on the approach, it is important to analyze the audio signal they produce on the touch surface. This analysis is achieved by doing a spectrographic analysis on each experimented gesture.

The analysis for 3 simple gestures is presented on Figure 3.4, Figure 3.5 and Figure 3.6.

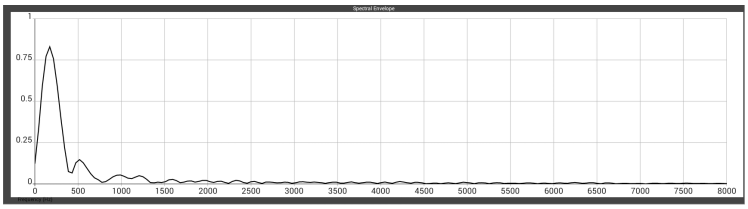


Figure 3.4: Spectrogram analysis for a tap gesture

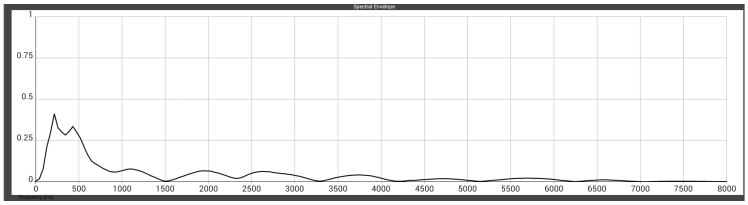


Figure 3.5: Spectrogram analysis for a nail gesture

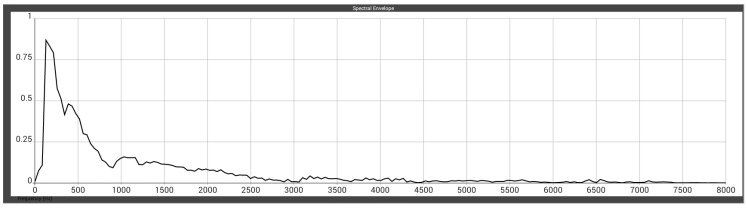


Figure 3.6: Spectrogram analysis for a knock gesture

These 3 gestures seem to be distinct enough, so these are the gestures considered on the approach.

Next, it is essential to focus on the audio filtering process. Although these gestures may be distinct and provide satisfactory results, a noisy environment may still greatly harm the success of the approach. However, using the adequate filtering strategies, information can still be retrieved and used. Filtering audio is a delicate task, as there is a fine balance between how much of the signal to be removed is relevant or not.

To ensure that the filters are applied with caution, a simple Android application was developed that presents the spectrogram of the processed signal, and allows for direct manipulation of the filters to apply.

The application was used on a relatively noisy environment (a coffee shop) and the 3 gestures already described were tested. On Appendix B some results of this analysis are

available. For example, the knock gesture (assumed to be the sharper one, with a higher peak than other experimented gestures), exhibits a fundamental frequency (the highest peak) of about 200Hz, but strong partials at 400Hz, 1200Hz and above.

Based on this analysis, the decision was to keep the range of frequencies from 100Hz to 2500Hz. This range is quite conservative, due to 2 main factors:

- We want to allow for some flexibility on the choice of gestures used on this approach approach;
- The captured signal may vary considerably depending on the device used and the environment on which it is used.

After filtering the signal, it is possible to start retrieving information.

This task is executed on the Pure Data patch, and so, the information processed is fully dependent on the dataflow programmed onto the patch.

This process is relatively simple: the signal is received, filtered and then passed to the audio processing objects (e.g. Bonk) where relevant information is retrieved.

This information is then sent to the toolkit and received by the AcousticListener component. This component receives the information from each beat detected by Bonk (in real-time) and parses the information which is then sent to AcousticClassifier to be analyzed.

This approach also includes the use of sound information to calculate gesture hit intensity on the surface, and classify this intensity for each gesture based on training data (described in the next section).

To achieve this, some alternatives were considered, all of them leveraging on features of Pure Data.

First we considered the simplest alternative: Bonk's loudness output. Already described in section 3.3, this output reports the overall loudness of the signal, obtained from the amplitude captured on all Bonk's filters.

The **[fiddle~]** object [P+98] performs real-time analysis of both signal amplitude and pitch, and is based on the FFT algorithm. The signal loudness is returned in decibels (dB). This is a rather simple alternative considering that information is analyzed at real-time, and can be returned together with Bonk's information.

Finally, a third alternative was studied: using Pure Data **[sigmund~]** object. This object provided the option to output a signal amplitude continuously. Using Bonk's beat detector, we can obtain the signal amplitude only when a gesture is detected.

Although the present work is not focused on creating a fully functional and efficient gesture intensity detector, we are going to evaluate these 3 alternatives and conclude on which of them yields better overall results.

3.2.4 Gesture Classification

With both the audio and touch events already acquired, it is possible to start the gesture classification process. This process is only possible if Bonk is already loaded with the training set of gesture classes.

Here the approach was to detect and classify gesture only if both touch and sound were detected (it is possible to receive sound information even if a touch did not occur, due to the difficulty on processing and filtering sound already described in previous sections), focusing on the touch event.

The gesture detection algorithm is presented on Algorithm 1.

Algorithm 1 Gesture detection algorithm

```
1: procedure GESTUREDETECTION
2:     Receive touch event T
3:     Receive sound event S
4:     if T.timestamp ≤ S.timestamp ≤ T.timestamp + latencyInterval then
5:         Gesture detected
6:     end if
7: end procedure
```

As Android currently possesses a latency problem, the solution was to detect if a sound event occurs from the moment the touch event is detected up until the latency interval (this value was set to 250ms, as explained on section 3.4).

3.2.5 Training Process

Our approach's classification feature relies heavily on the acquisition of data to train the classification algorithm.

The training process if fairly simple while using the audio processing tool (e.g. Pure-Data) as a standalone PC application. To successfully train a gesture to be classified, 10 samples of each gesture must be provided. This information will be processed and recorded, and the classification process is therefore ready to be used with success.

To incorporate this process on other devices, while maintain the specificities of the training process, a device-specific application is required.

This happens mainly because the training process is heavily dependent on the type of surface used, and the technical details of the device, such as operating system, signal processing capabilities and reliability of microphones (either built-in or external).

As our approach includes both small touchscreen devices and wider tabletop surfaces, two separate training applications were developed. The tabletop training application is described on subsection 3.2.7. Although the concept and main features are similar, the mobile device application provides more advanced features, to better integrate the training process with our mobile device's development toolkit.

On Figure 3.7 the mobile device training application is presented.

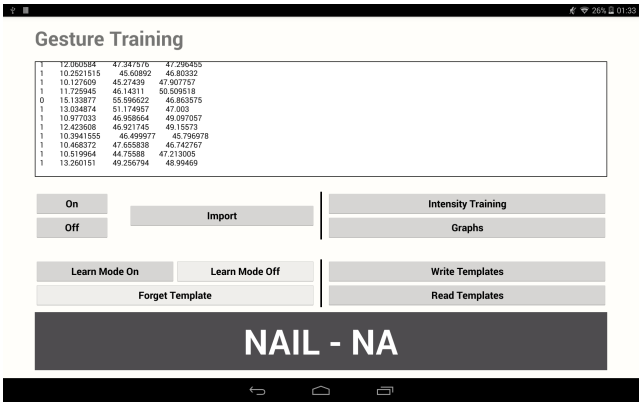


Figure 3.7: Mobile device training application

This application provides all features needed to train gestures and import previously trained sets:

- Control the gesture training process (start and stop or forget part of the set);
- Train intensity levels for each gesture;
- Name each gesture to be used on the classification process;
- Import and export the training set from and to external files;

The import and export features are necessary to integrate the toolkit into other applications (more details on subsection 3.2.6). The application's source code is available on GitHub [1]. The intensity training process is independent from Bonk's gesture training process. The intensity training application is presented on Figure 3.8.

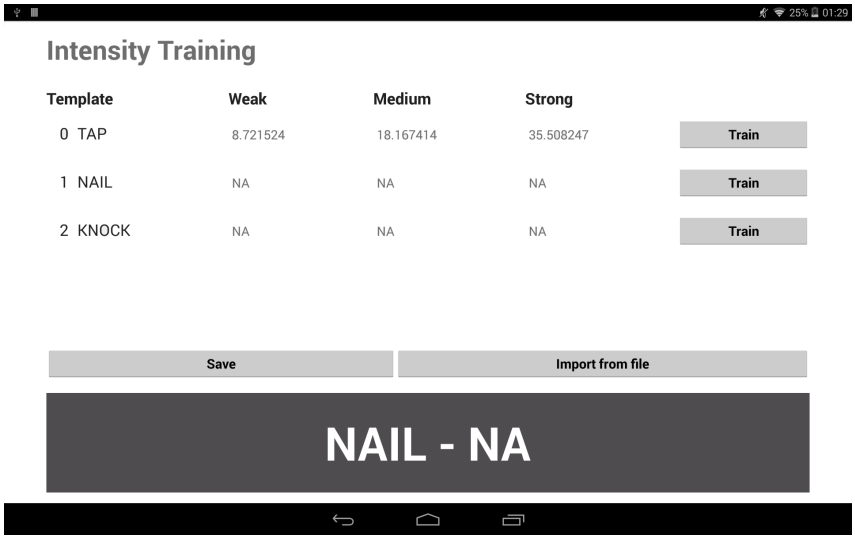


Figure 3.8: Mobile intensity training application

[1] https://github.com/PedroSoldado/RecognizerEvaluation

There are three intensity levels: weak, medium and strong. To train each one of these levels for each gesture, the user repeats each gesture 15 times, 5 times for each intensity level. The intensity level is then calculated by simply averaging these 5 values.

This training set can also be saved to a file and reused later.

3.2.6 Integration with applications

With all the toolkit's internal processes explained, the integration structure can be described.

One of the toolkit's main objectives is to provide gesture classification through sound features in a complete, yet simple and efficient manner. This of course has an impact on how complex the integration process must be for an application developer.

The first concern is related to the amount of additional coding needed to introduce these new features into a new or even already developed application. This additional effort must be cut to a minimum, for the toolkit to be used.

To successfully use the toolkit's features in another application, it is first needed to include the library into the application's development project, to allow access to the components previously described.

Next, some code modifications must be applied. The code modifications needed to execute the toolkit's gesture classification on an Android application are presented on Algorithm 2 and Algorithm 3.

Algorithm 2 Android activity initialization

```
procedure CreateActivity
    Initialize activity
    Initialize a PureDataRecognizer instance
    ...
    Attach listener to recognizer
    Add sound information to recognizer
    ...
end procedure
```

Algorithm 3 Android touch event handler

```
procedure DispatchTouchEvent
    Process touch event
    ...
    Add touch event information to recognizer
    if Gesture detected then
        Get gesture name
    end if
    ...
end procedure
```

The pseudocode lines on bold correspond to the integration with the application. An actual example of implementation on an already functional application is presented on Figure 3.9 and Figure 3.10.

```java
public static PureDataRecognizer recognizer;
public static PureData pd;

@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);

    recognizer = new PureDataRecognizer(getApplicationContext());
    pd = new PureData(getApplicationContext());

    pd.getMyDispatcher().addListener("bonk-cooked", new PdListener.Adapter() {
        @Override
        public void receiveList(String source, Object... objects) {
            int template = (int) Double.parseDouble(objects[0].toString());
            float velocity = Float.parseFloat(objects[1].toString());
            float colorTemperature = Float.parseFloat(objects[2].toString());
            float loudness = Float.parseFloat(objects[3].toString());

            recognizer.addHit(template, velocity, colorTemperature, loudness);
        }
    });
}
```

Figure 3.9: Toolkit Integration - Class initialization and sound event handler

```java
if(event.getAction() == MotionEvent.ACTION_DOWN){
    recognizer.addTouch(xPosition,yPosition);
}

String gesture = recognizer.getGesture();

switch(gesture){
    case  "TAP":
        doA();
    case "KNOCK":
        doB();
    case "NAIL":
        doC();
    default: break;
}
```

Figure 3.10: Toolkit Integration - Touch event handler and gesture-to-feature mapping

On Figure 3.9 it is exemplified how to start the toolkit's features on the Android activity where it will be first executed. The toolkit's main entry point is the Recognizer class, which uses an auxiliary class, PureData.

Then the sound event listener component must be initialized. This initialization uses an adapter to retrieve only the needed information. Although there is the need to add these lines on the application's side, there is no need for additional configurations (this explicit listener implementation cannot be avoided due to the need to explicitly attach the listener to the Android background service already running on the toolkit side).

Figure 3.10 exemplifies the touch event handler. When a touch is detected, its location must be added to the toolkit. The toolkit is then ready to be queried for a classification of the used gesture, resulting on the mapped name of the gesture. The mapping of this gesture name to a feature on the application is fully dependent on the developer's choice, but a simple example is showcased.

The open-source code is published on GitHub, and accessible through the toolkit's web-

site [2]. These changes and the impact on the resulting developed application (at a technical level) are going to be evaluated on chapter 4.

3.2.7 Tabletop Toolkit

The tabletop toolkit is a simpler version of the one developed for mobile devices. The objective here is to study if our approach can be applied to other touch-enabled surfaces, and create applications that use it to enhance interaction.

Although this toolkit was designed to work on a wide variety of tabletop surfaces, it is important to describe the setup with which it was used and tested.

The tabletop is composed by a computer connected to a monitor, with a touch-sensitive overlay on top of it and also connected to the computer.

To capture the sound information, a pickup was used. These devices are commonly used to tune musical instruments, as they are very efficient at cancelling ambient noise. The pickup was attached to the corner of the tabletop, and connected to the computer. A AcousticProcessing instance is executed on the computer, with the same patch used on the mobile device's toolkit.

On this toolkit the information is passed to the application via a socket (Pure Data supports this out of the box), and the sound event is received and associated with a touch event (received from the touch overlay). Due to this fact, this toolkit is very simple, and integrating it with other applications only requires the implementation of a information receiving component (to receive the classification result from the Pure Data patch).

The training process is made directly with Pure Data, and a subpatch was developed to assist in this process. This way the training set is directly trained onto Bonk.

This subpatch is presented on Figure 3.11. With this simple patch, it is possible to start and stop the training process, present the number of samples trained for the current gesture, and analyze the spectrogram of the incoming sound.

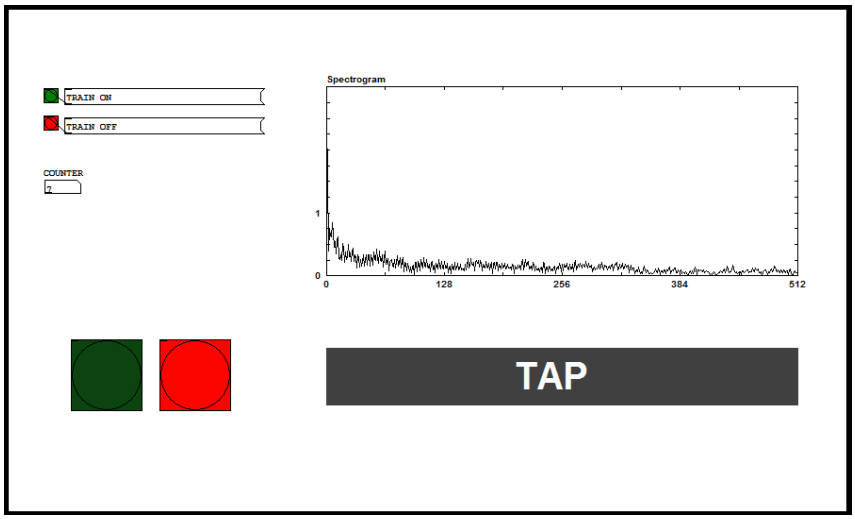


Figure 3.11: Tabletop training subpatch

[2]http://web.tecnico.ulisboa.pt/ist170184/impactToolkit

The gesture detection algorithm is much simpler than the one used on the mobile device's toolkit, as each sound event strictly corresponds to a touch event (ambient noise is cancelled almost entirely).

All these architecture details and implementation are supported by a set of considerations that were studied and analyzed during all this work. These considerations are presented on the final section of this chapter, on section 3.4.

As already described, Bonk represents a central part of the approach, providing both audio processing features, and also machine learning capabilities, so it is important to describe this object in further detail to understand how it works, and how it is incorporated on this architecture.

3.3 Bonk

The [bonk~] tool (simply referred to as Bonk [P+98]) is one the main components of this approach. Bonk is a Pure Data object used to detect beats and patterns given an audio signal. It is, at its core, a beat detection tool, as its main output is a simple and raw analysis of the attack detected (abrupt change on the incoming signal).

However, leveraging on the other output Bonk produces, it can also be used as a machine learning algorithm [MRT12], to classify new information based on previously acquired (trained) data.

Next, a brief explanation will be provided on on how Bonk operates, what outputs it provides, and its main features.

3.3.1 Technical Details

Bonk analyzes incoming signals (received from the audio input channels) using a FIR filter bank (an array of band-pass filters) composed of 11 filters, with 2 filters per octave (as suggested in [P+98]). Each of these filters carries a sub-band of the original signal.

An overview of Bonk's architecture (simplified) is presented on Figure 3.12.

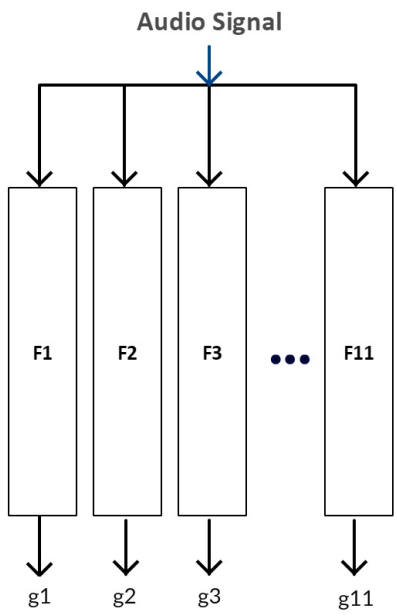


Figure 3.12: Bonk Architecture Overview

Bonk is basically a set of 11 filters, each one of them responsible for part of the signal processing. The processing algorithm is explained below.

Bonk's analysis results in 2 outputs:

1. A raw analysis of the audio signal, represented by 11 values, one for each of the filters. This output represents the whole spectrum of the attack;

2. A number, representing the best-matching template for each attack, and the overall loudness of the attack.

Beat Detection Algorithm

The attack detection algorithm works best at detecting percussive attacks (such as hits on a drum set, or on a surface). To achieve this, Bonk relies on the fact that a percussive attack on a surface often represents an abrupt change on the audio signal that is produced around that surface.

To measure these changes, and identify an attack, Bonk applies a growth function to each channel. This growth function simply compares the current power (calculated as defined in [P+98], and illustrated on Figure 3.12) in the channel (filter) to a saved value: if current power is higher than the saved value, the current one is saved. The growth is then calculated as follows:

$$g = max(0, p/m - 1)$$

where m is the mask value (saved value) and p is the current power in the channel.

The growth values for all 11 channels is added up, and if the sum is greater than a given threshold (12 for this algorithm), an attack is reported.

Classification Algorithm

To analyze real-time data and classify it, Bonk employs a machine learning approach. In particular, it uses a supervised learning strategy to achieve classification capabilities. This strategy specifies that all categories (in our case, the gestures) must be known beforehand and provided in advance to the algorithm (these categories are commonly referred to as a training set).

This training process must be the first task to be executed to use Bonk. The Bonk help manual recommends that each template should be trained 10 times, to improve successful classification chances.

After training, Bonk is ready to start receiving new data. This new data is captured by Bonk, and the spectral template is extracted as explained before (using the growth function). This template is then matched against the training set, and the best match to the current training set is reported.

In the next subsection we will describe how Bonk is used on our approach.

3.3.2 Use On the Approach

After a brief explanation of Bonk's characteristics and features, it is simple to understand why it was used on our approach.

First, it is part of Pure Data, a relatively simple yet powerful tool to process signal (specially audio signal), full of interesting features. Bonk comes as a nice addition to these features, providing the machine learning and classification components needed for building our approach.

In our approach, we use Pure Data mainly to filter the incoming signal, and to tune Bonk's parameters. Then, both of Bonk's main features are used on our approach.

It uses the beat detection algorithm to detect attacks and our own algorithm (explained later) saves these attacks. Some of the attacks are then compared to our training set.

3.4 Additional Considerations

To complete the description of the approach developed, there are some additional considerations to take into account.

As explained before, Android currently suffers from a audio latency problem. This factor lead to some intensive study to prevent this latency from harming the gesture classification process.

As this problem naturally extends to the approach, and to how Pure Data receives the audio signal, an analysis was executed to decide if this represents a critical problem to our approach, we used a simple Android application called AndroidLatency [Pur], available at

the Pure Data official website, to analyze the audio latency for the setup used for developing our approach (tablet BQ Aquaris E10).

The latency (round-trip latency) value obtained was about 500ms for this device, and it is also the average value for other devices as well. Considering the recording latency (time between the signal entering the input until it is processed by the device) to be half of this value, the final value is 250ms of latency. This value was included in the approach, on the gesture detection algorithm as already presented.

This problem was not observed on the interactive tabletop side of the approach, because of the the added reliability provided by the use of an external microphone. This fact also greatly reduced the need to filter sound while developing the tabletop toolkit.

Chapter 4

Evaluation

After developing the approach, it is important to evaluate all the processes and main features, to study the results and conclude if the research statements previously defined were successfully answered.

This chapter will focus on all these evaluation and testing executed on the project. The main results are also described.

It is divided in two main parts, one for the mobile implementation of the toolkit, other for the tabletop implementation.

The main evaluation task is to evaluate the gesture classification process, because a positive result from this tests allows to answer the first research question defined for this work, regarding the development of an approach of gesture classification through acoustic sensing.

To answer the second research question, related to the improvement of user's interaction with touch applications, extensive user testing and application testing is also going to be executed.

Regarding the mobile version, the following additional evaluations will be executed:

- User testing with the prototype application, to validate if the interaction obtained from the developed approach are satisfactory and applicable to other applications;
- The toolkit performance while integrated on an application, at a technical level, to study the memory and CPU processing requirements, and the impact on targeted applications;
- Sound intensity classification evaluation, to validate if this part of the approach may also be used to improve the toolkit's features.

The tabletop toolkit will be the target of additional testing, by using another prototype application, integrated with the toolkit, because it is important analyze the level of interaction achieved by using gesture classification on these kind of devices.

4.1 Mobile Devices Evaluation

The mobile device's toolkit is the main component of the project. It is a simple yet robust set of components that provide gesture classification features by analyzing the sound produced with interaction with mobile devices. The evaluation aspects must comprehend all aspects of the work developed on this project.

The following sections describe the methodologies used and results from this evaluation.

4.1.1 Gesture classification Test

The most important metric to evaluate on a gesture classification toolkit is the effectiveness of the classification process. This is accomplished by training the gesture set with the gesture classifier, until the results are considered solid and representative enough, and then test the classifier against a set of the same gestures. This test was executed by the same user for all test sessions, on the same environment (noise-free, calm environment), and on the same device, to guarantee that all results were retrieved on the same conditions.

Evaluation Methodology & Results

To evaluate the classifier, a confusion matrix [Faw06; Ste97] was used. This methodology compares expected results (for each gesture trained into the algorithm) against actual results and presents them in a matrix, which allows useful analysis to be done on the results obtained (for example, the overall accuracy of the classifier can be easily calculated). It is a simple yet very efficient method used mainly to evaluate the performance of algorithms, specially supervised learning algorithms.

The gestures considered for this test, and a brief description of each of them are presented on Table 4.1.

Name	Description
TAP	A simple tap on the surface, equal to the gesture used while interacting with a touch device
NAIL	A tap using a nail to hit the surface
KNOCK	A knock on the surface, similar to a knock on a door
STYLUS	A tap with a capacitive stylus pen

Table 4.1: Mobile Test - Gestures name and description

The test was carried on as follows: each gesture was tested 20 times, and each result returned by the classifier (the algorithm's classification for the tested gesture) was written down. After gathering all the results, the confusion matrix was built. The matrix is presented on Table 4.2.

		Predicted Class			
		TAP	NAIL	KNOCK	STYLUS
Actual Class	TAP	20	0	0	0
	NAIL	0	18	1	1
	KNOCK	1	0	17	2
	STYLUS	5	1	3	11

Table 4.2: Confusion Matrix - Mobile classification test results

This preliminary results already provide some conclusions:

- The TAP gesture (the simpler and yet more familiar gesture) was correctly classified for all 20 samples;
- The NAIL and KNOCK gestures provide sufficiently stable results to allow an efficient interaction;
- The STYLUS gesture introduces entropy on the classifier, as some of the other gestures were incorrectly classified as STYLUS.

Using the data on the confusion matrix, a table of confusion was created for each gesture. A table of confusion [Faw06] (illustrated on Table 4.3) is a table with 2 rows and 2 columns, used to report the number of **false positives (FP)**, **false negatives (FN)**, **true positives (TP)**, and **true negatives (TN)**.

The resulting table of confusion for the TAP gesture is presented on Table 4.4, and the complete analysis can be consulted on Appendix C.

		Predicted Value		
		p	**n**	**Total**
Actual Value	**p′**	True Positive	False Negative	P′
	n′	False Positive	True Negative	N′
	Total	P	N	

Table 4.3: Table of confusion

		Predicted Value		
		p	**n**	**Total**
Actual Value	**p′**	20	0	P′
	n′	6	54	N′
	Total	P	N	

Table 4.4: Table of confusion for the TAP gesture

It is simple to understand the table: for example, the number of false positives is 6. This means that 6 of the tested gestures were incorrectly classified as a tap.

With all this information organized, the accuracy for each gesture was calculated. Accuracy is a statistical measure used to describe the closeness of a result (or group of results)

to a standard or expected value, being widely used to conclude how well a classification algorithm performs.

Using the data calculated on the table of confusion, the accuracy can be calculated as follows:

$$accuracy = \frac{TP + TN}{Positive + Negative}$$

The accuracy for each gesture is presented on Figure 4.1.

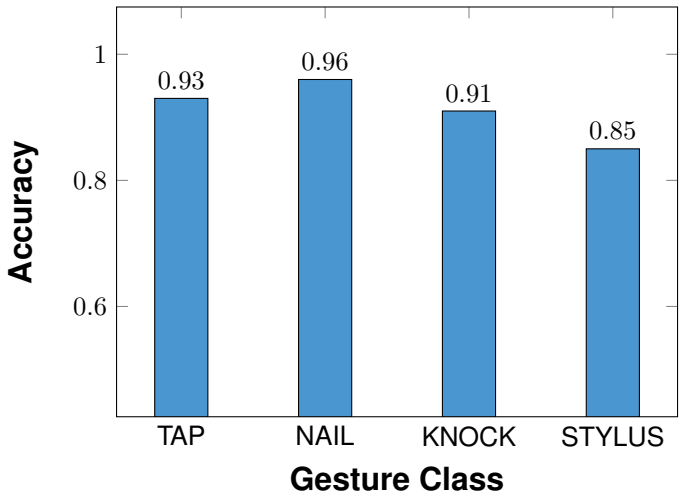


Figure 4.1: Mobile gesture classification accuracy results

The overall accuracy (considering all 4 gestures) of the recognizer was approximately 91%.

Results discussion

The results obtained prove that the approach developed can be considered as another alternative for designing user interaction, and be explored and extended to improve navigation on user interfaces.

Another important conclusion is related to the set of gestures that can be used. As observed, the addition of a relatively different gesture (in this case, the STYLUS gesture, applied with an object) can cause entropy on the results, and harm classifier performance. By removing this gesture, and applying the same methodology, the overall accuracy would have increased by approximately 5%.

The advice taken from these results is to choose only strongly representative and expressive gestures, that can be unequivocally distinguished. Taking this advice into account, the STYLUS gesture will not be considered on the user tests (described on subsection 4.1.3).

4.1.2 Toolkit Integration Test

To validate that our approach can be utilized by other developers to leverage on all the features it provides, and that it is integrated in a simple and efficient manner on application development life-cycle, it is important to test this toolkit with the main stakeholders, i.e., application developers.

To execute these tests, the collaboration of 2 application developers was requested. To speed up the process, a sample application (not integrated with the toolkit) was provided to both developers. This application is the same one used throughout all testing sessions, i.e., the painting application, further described on the next section. The developer was asked to import the application's Android project, and to start the integration, following the process described on the toolkit's GitHub repository[1].

The metrics captured for both development sessions were:

- Time taken until the integration was successfully completed (including time to test it on a mobile device);
- Number of lines of code to successfully integrate the toolkit.

Metric	Time to Completion	Lines Of Code	Difference on Lines of Code
Baseline	1 hour 10 minutes	25	-
Developer #1	2 hours 10 minutes	25	0
Developer #2	1 hours 35 minutes	17	- 8

Table 4.5: Integration test results

The first row of the table presents the result for a test executed by the toolkit developer, to create a baseline to compare the integration test results.

The second row presents the results for the first developer, a developer with very little experience on application development. It can be observed that, comparing to the baseline value, it took him an additional hour to integrate and test the toolkit on the sample application. However, the lines of code utilized are the same as the baseline value, because the integration guidelines were strictly followed.

The second developer was an experienced application developer, which reflected on the time taken to integrate: a difference of only approximately 25 minutes from the baseline value. It is also interesting to observe that this developer optimized the integration process on the number of lines of code used, with a difference of minus 8 lines from the ones used on baseline test (the difference comes from the direct use of variables retrieved from the sound event handler, without the need to first declare a new local variable).

Comparing both results to the baseline value, it is possible to conclude that the toolkit's integration process is executed on a reduced number of hours, which will hardly cause any delay on the development process. The added complexity is also not a problem, as the main changes are the initilization of the toolkit's features, and the touch event handler explicit declaration, with no additional loops or complex statements added.

[1] https://github.com/PedroSoldado/RecognizerEvaluation

4.1.3 User Testing

This work would not be complete without testing the successful integration of the toolkit developed, with an application.

To prove it, a painting application for a tablet device was developed. This application allows free hand drawing with current touch interaction, with options presented as buttons and menus. The toolkit was successfully integrated into the application, by applying the changes previously described on chapter 3.

This new application was named ImpactPaint, and its user interface is presented on Figure 4.2.

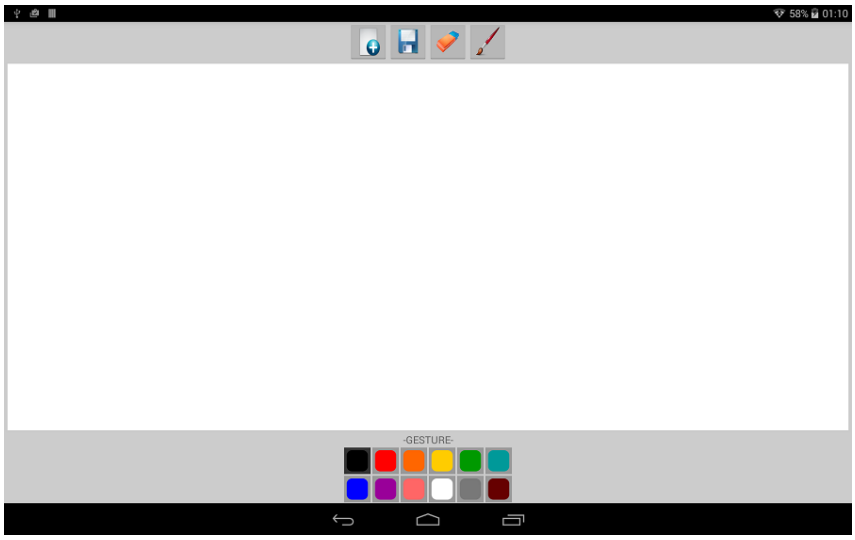


Figure 4.2: Integrated application - ImpactPaint

It provides a simple set of features: brush draw, eraser, saving to file, and a color picker. The integration mapped some of these features to gestures:

- TAP - a simple tap on the surface to simply draw on the canvas, and set brush color to black;
- NAIL - a nail tap, which activates the eraser;
- KNOCK - by knocking on the surface, to toggle a circle drawing brush.

Next, the users tests were run using this application as example.

Evaluation Methodology

For the user testing, a population of 15 users was considered, with ages between 18 and 50 years. All users are comfortable with applications of this kind, and are capable of using mobile devices such as smartphones or tablets.

The main objective of these tests is to validate if the work developed on this approach really allows for another level of interaction with user interfaces, and if such interaction is executed in an efficient and simple manner.

To study this hypothesis, each user was asked to perform a set of tasks that use the gestures mapped on the application (the user test guide is presented on Appendix D).

The tasks to be executed are:

1. Draw a ”#” symbol on the screen with taps;
2. With a nail tap activate the eraser and erase one of the lines, and draw another line;
3. Knock on the screen to activate the circle brush and draw 2 circles, knock again to deactivate the circle brush, activate the eraser and erase one of them, and draw a straight line over the remaining circle;

These tasks were designed to showcase the available gestures, and the flow of interaction of an application of this type.

While the user executed each task, the metrics collected were: time of execution, number of errors and the number of actions to accomplish each task. An action here corresponds to the number of gestures, successfully applied or not, to finish the task, i.e., this metric takes into account the total number of actions executed by the user (including simple navigation, or interaction with other aspects of the applications, such as buttons or options not relevant to the task being executed).

These metrics allow a complete analysis of the user’s experience. The time of execution is estimated to grow with each task, as the complexity also is greater from task to task. The number of errors here reflect the reliability of the gesture classification process, as a high number of errors corresponds to a high number of wrong classification results (as the user does not obtain the expected result from the applied gesture).

The number of actions provides input on how the user perceives the interaction with the application and the effort to execute each task.

In the end of the testing session, a simple survey was provided to each user. The objective of this survey was to receive feedback from the user’s experience and satisfaction towards the approach.

Results

The results of all the testing sessions were analyzed in order to obtain a quick overview of the users’ experience. These results are condensed on Table 4.6.

Task	**Time (seconds)**		**Errors**		**Actions**	
	Average	**Standard Deviation**	**Average**	**Standard Deviation**	**Average**	**Standard Deviation**
1	4,90	1,56	1,33	0,99	5,93	1,22
2	10,17	1,60	2,60	1,40	6,13	1,81
3	20,74	6,32	3,13	1,64	10,07	1,94

Table 4.6: User testing results

As expected, tasks were executed on the estimated time, growing on execution time (due to the growing complexity of the task). The third task presents a high standard deviation due to the confusion of one of the users concerning the sequence of gestures to apply. This factor can be explained by the lack of familiarity to an approach of this type.

The number of errors is relatively low, and can be related to the fact that users did not have previous experience with this approach to gesture classification, or the prototype application's features. However, as the task complexity grew, and more gestures were needed, the number of errors did not escalate further than expected, so it is possible to conclude that the learning curve is not too steep for everyday use.

The number of actions corresponds to what was expected, with a growing number of actions as the task grew more complex. The standard deviation value confirms the steady behavior from all users (no perceivable variation on the number of actions expected from each of the users).

The survey consisted on a question regarding the user's satisfaction and interest in using more applications developed with this approach, and a space for comments or suggestions. The question was graded from 1 to 5, with 1 being "No interest, would not use", and 5 being "Very interested, would use". The results are presented on Table 4.7.

Answer	**1**	**2**	**3**	**4**	**5**
Totals	0	1	3	6	5

Table 4.7: User Testing - Survey results

The results are overall very positive, with a majority of the users (11 of the 15 users) considering this type of interaction as interesting, and available to use it on other applications.

The user that answered "2 - Not very interested, might not use", commented that it was confusing to the user to know what gestures to apply, as no visible help was provided. This feedback can be taken into account to future developments.

4.1.4 Toolkit Technical Evaluation

The approach developed focuses mainly on how user interaction can be enhanced with gesture classification features. However, one of the main objectives is to provide these features without harming the performance of the target application.

For performance measurement, the resource utilization of the ImpactPaint application is going to be studied. The metrics to be studied are the CPU variations and the memory requirements of the application performing while with or without the integration of the toolkit. These metrics can be directly retrieved from Android Studio debug logs.

CPU Performance

On Figure 4.3 and Figure 4.4 the CPU performance differences are presented.

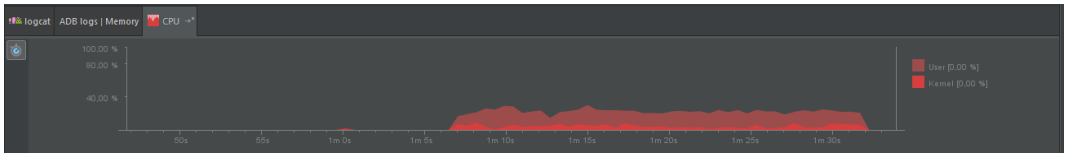


Figure 4.3: ImpactPaint CPU performance without the toolkit

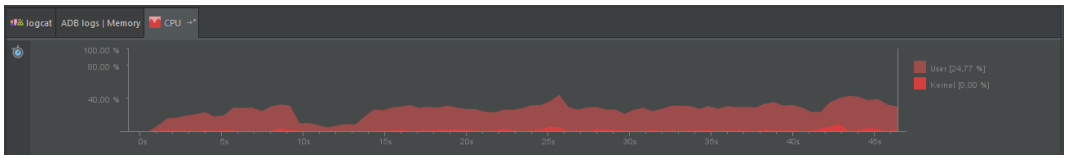


Figure 4.4: ImpactPaint CPU performance with the toolkit

The CPU utilization slightly increases while the application is being used (when the toolkit is processing information and sending it to the application). However, there are no visible peaks of CPU activity and no sharp increase was identified, so the conclusion is that the toolkit can be successfully integrated without compromising the target application's performance.

Memory Performance

On Figure 4.5 and Figure 4.6 the memory performance differences are presented.

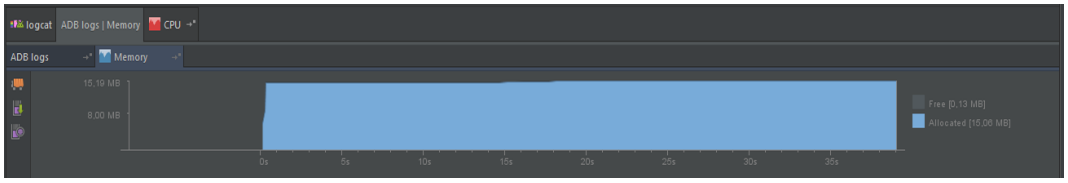


Figure 4.5: ImpactPaint memory performance without the toolkit

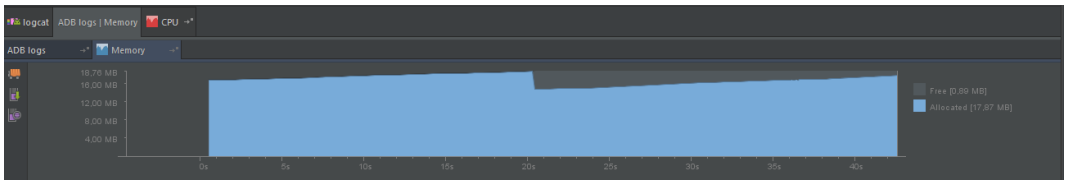


Figure 4.6: ImpactPaint memory performance with the toolkit

Without the toolkit, the application uses approximately 15MB of memory, and no great variations are observed while using the application. By integrating the toolkit, the memory requirements oscillate between 15MB up to approximately 18MB while using the application (when the toolkit is processing information).

Results Discussion

Both the CPU and memory performances are not compromised by integrating the toolkit in the target application, with no visible sharp changes on the performance when the toolkit was added.

However, some more advanced tests could be executed, in particular for different utilization rates and different types of applications.

4.1.5 Intensity classification Test

To study the possibility of obtaining and classifying gesture intensity levels based on the sound produced when they hit the surface, three alternatives were considered and tested.

This section presents the results obtained on this testing session, and the main conclusions drawn from this hypothesis.

For each alternative (already described on previous sections), the testing methodology is identical.

On this test session, only the TAP gesture will be considered, as it is sufficient to take conclusions. The toolkit is configured to receive the desired information as the actual intensity of the gesture. Then, the classifier is trained with the three intensity levels (using the training application).

Finally, using the training application, each intensity level is tested 10 gestures, and the result obtained is written down.

Bonk loudness

The first alternative to consider was the simpler one: leveraging on the output provided by Bonk, which includes the loudness of the audio signal.

The results for this alternative are presented on Table 4.8.

	Intensity Level		
	Weak	Medium	Strong
Yes	9	6	8
No	1	4	2
Total	10	10	10

Table 4.8: Sound intensity classification - Bonk's loudness

For each of the intensity levels, the percentage of successful classifications are satisfactory: 90% for the ”weak” level, 60% for ”medium”, and 80% for the ”strong” intensity level. Overall, the accuracy obtained is approximately 77%.

Amplitude tracking

This alternative uses amplitude tracking to estimate gesture intensity. This is achieved by passing the audio signal through the Pure Data's [sigmund~] object.

We capture the amplitude output when an attack is detected, and it is sent to the toolkit. The results are presented on Table 4.9

	Intensity Level		
	Weak	Medium	Strong
Yes	6	4	7
No	4	6	3
Total	10	10	10

Table 4.9: Sound intensity classification - Amplitude

With this alternative, a value of approximately 57% average accuracy was obtained.

It is important to observe that, although the results for the ”weak” and ”strong” intensity levels are quite positive, the ”medium” level was correctly classified only 40% of the times. This occurs because the amplitude value retrieved from [sigmund~] ranges from 0 to 1, but the experimental values for the captured audio signal start at about 0.5, giving a small range for the classification process to distinguish the levels in an effective way.

Loudness

The third alternative uses [fiddle~] Pure Data object, mainly used to estimate pitch and amplitude of incoming sound.

The results for this alternative are presented on Table 4.10

	Intensity Level		
	Weak	Medium	Strong
Yes	5	3	8
No	5	7	2
Total	10	10	10

Table 4.10: Sound intensity classification - Loudness

For this alternative the classification process was correct for approximately 53% of the tested gestures.

Overall, the results are as follows:

	Alternative		
	Bonk loudness	Amplitude tracking	Loudness
Success Ratio	77%	57%	53%

Table 4.11: Overall intensity classification results

It is safe to conclude that the best alternative is, after all, the simplest one. Using the output provided by Bonk, and simply calculating the average of 5 training examples, a 77% accuracy ratio was obtained.

However, there is something that must be considered: audio processing is a relatively complex task, where a small difference on the time at which the signal is captured can harm the results. The approach developed may not guarantee synchronization on all audio processing tasks, and so the results obtained from the [sigmund~] and [fiddle~] objects may not be exactly synchronous as the ones obtained from Bonk (being the same object that detects the gestures).

Although a successful classification ratio of 77% is positive, it may not be sufficient to use this implementation to classify interaction gestures, at the risk of impairing and adding unnecessary entropy to user interaction.

4.2 Tabletop Evaluation

The main idea behind this test session was to understand if our approach to using acoustic sensing as a way of interaction with interfaces could be applied to wider surfaces, and if new gestures and dimensions could be exploited by the additional interaction space provided by these surfaces.

To achieve this, some gestures used on the mobile version of the toolkit were reused and trained using the table. Furthermore, some more complex gestures were tested on the table: a punch, a palm hit and a pen tap.

Leveraging on the fact that the touch technology used by the table is different from the one on most tablets (resistive touch overlay instead of capacitive displays), other objects can also be used to interact with the table's surface. To achieve this, common objects were tested against the surface: an eraser and a pen (tip of the pen cap). The results obtained with the eraser were not clear enough to allow for a clear classification, but the sound produced by the pen hitting the surface was clearly distinct, so this gesture was also considered.

4.2.1 Gesture classification Test

The gestures considered for this test, and a brief description of each of them are presented on Table 4.12.

Name	Description
TAP	A simple tap on the surface, equal to the gesture used while interacting with a touch device
KNOCK	A knock on the surface, similar to a knock on a door
NAIL	A tap using a nail to hit the surface
PUNCH	A punch with closed hand, hitting vertically
PALM	A PALM with open hand, hitting with the palm of the hand
PEN	A hit with the tip of the pen cap

Table 4.12: Tabletop Test - Gestures name and description

This test was executed on a tabletop surface, by just one person, on a calm and noise-free environment.

The setup used was composed of a tabletop with a touch display, connected to a PC. Additionally, an external audio-capturing device (a pickup transducer) was used. This device was placed at the table corner just right of the user, in contact with the tabletop external case.

To first train the classifier, 10 samples of each gesture were used, with 60 samples in total. The already developed Pure Data subpatch was used on this training session. To guarantee the best possible set of gestures for the testing session, the algorithm was trained 10 times.

To test the tabletop gesture classifier, the same methodology used on the mobile implementation of the toolkit was used: a confusion matrix was built for the results, and then a table of confusion was created for each gesture.

The test was carried on as follows: each gesture was tested 20 times, and each result returned by the classifier (the algorithm's classification for the tested gesture) was written down. After gathering all the results, the confusion matrix was built. The matrix is presented on Table 4.13.

		Predicted Class					
		TAP	KNOCK	NAIL	PUNCH	PALM	PEN
Actual Class	TAP	20	0	0	0	0	0
	KNOCK	1	18	1	0	0	0
	NAIL	2	2	16	0	0	0
	PUNCH	1	3	0	14	2	0
	PALM	0	1	0	2	17	0
	PEN	0	0	0	0	0	20

Table 4.13: Confusion Matrix - Tabletop classification test results

Before further analysis, there are some quick conclusions that can be taken:

- Both the TAP and PEN gestures were correctly classified on all 20 samples;
- The TAP and KNOCK gestures were confused with other gestures the most;
- The PUNCH gesture was the least successfully classified (only 14 out of 20 times).

Using the data on the confusion matrix, and following the same methodology, a table of confusion was created for each gesture.

The resulting table of confusion for the TAP gesture is presented on Table 4.14, and the complete analysis can be consulted on Appendix E.

		Predicted Value		
		p	**n**	**Total**
Actual Value	**p′**	20	0	P′
	n′	4	96	N′
	Total	P	N	

Table 4.14: Table of confusion for the TAP gesture

Next, the accuracy for each gesture was calculated, so more specific conclusions can be taken from this evaluation.

The accuracy for each gesture is presented on Figure 4.7.

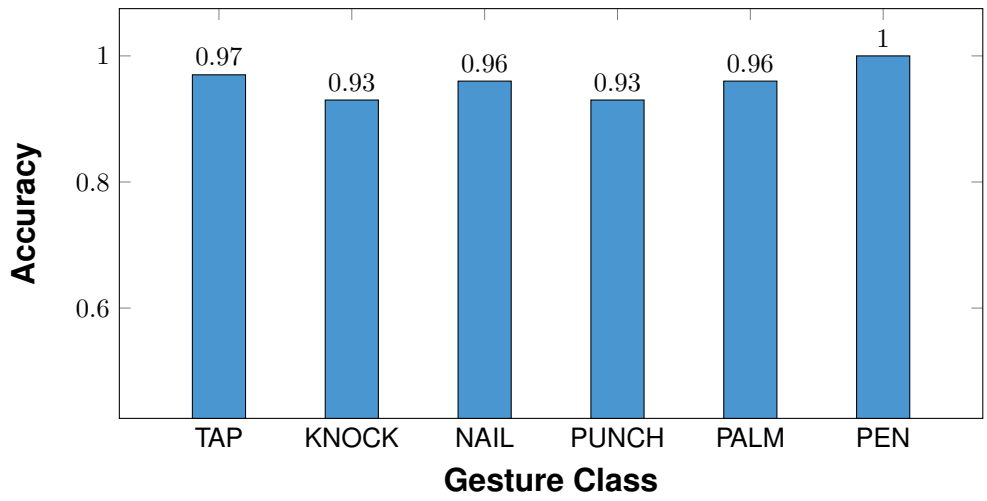


Figure 4.7: Tabletop gesture classification accuracy results

The overall accuracy (considering all 6 gestures) of the recognizer was approximately 96%. This is a very satisfactory result, and proves that this gesture classification approach can be successfully applied to tabletop surfaces.

Another important aspect was to understand if the location of the interaction on the surface (closer to the pickup or further away) could affect the classification process.

To test this hypothesis, the tested gesture must be sufficiently complex (e.g. use a wider portion of the body) to allow for some fluctuation of the results.

Considering this factor, the PUNCH gesture was selected, as it uses the whole side of the hand to hit the surface.

To run this test, the touch table was divided into 3 zones: a zone close to the microphone, the middle of the table, and the opposite side of the table. This separation was based on previous empirical results (this hypothesis was raised while training the classifier).

	Distance to Pickup Microphone		
	Close	Middle of Table	Opposite side
PUNCH	9	7	4
Other	1	3	6
Total	10	10	10

Table 4.15: Hit proximity test

This results prove that the location of the gesture on the surface can harm gesture classification, if only one sound capturing device is used. The solution to this problem might be to by simply adding more audio-capturing devices to the surface. This is not considered on the toolkit's implementation, but the main alteration would be to consider this additional audio input, and process each classification result separately, and then comparing them, pondering each result with the distance at which the gesture was executed (detected by analyzing the touch event location).

4.2.2 Application Test

To ensure that the tabletop version of the toolkit also yields positive results on a more complete experience (when the toolkit's features are used on another application's interaction), a prototype application was developed to showcase these features.

The main objective of this evaluation is to validate the results obtained on subsection 4.2.1, and observe if a full interaction with a fully interactive application can be achieved. However, it is not on scope to test this application with other users or to undergo a more complete test and analysis of this application. Due to this fact, the results are a set of empirical conclusions and observations.

This test was executed in the same conditions of the previous section, by only one user, the developer of the toolkit.

This new application was named PhotoManager, and its user interface is presented on Figure 4.8.

Figure 4.8: Integrated application - PhotoManager

This application allows the manipulation of pictures. On Table 4.16 the mapping of gestures to application's actions is presented.

Name	Description
TAP	Basic movement
KNOCK	Double the size of the picture
NAIL	Create a copy of the picture
PUNCH	Remove the picture
PALM	Reduce to half of the size
PEN	Change the picture

Table 4.16: Tabletop Application Test - Gesture Mapping to Features

The tests consisted on a free interaction with the application and its features.

Overall interaction was very fluid and efficient, as all the used gestures were correctly classified and mapped to their features. This empirical analysis corroborates the results obtained on previous testing, and guarantees a positive answer to both research questions, as the integration process is reduced to initializing the toolkit's services and detecting touch and sound events, and interaction achieved with the toolkit is fluid and efficient.

4.3 Discussion

By analyzing all of the evaluations results, the developed approach can be considered a success.

The gesture classification tests yielded very positive results, providing a strong positive answer to the first research question. The developed toolkit (the mobile and tabletop versions) correctly classified the trained gestures over 90% of the times.

The user testing and application navigation tests corroborate the hypothesis described on the second research question. Not only this approach allows a simple development and integration with other applications, it also allows a satisfactory level of interaction with the application (with over 70% of the tested users being satisfied with the experiment and the interaction achieved).

However, it is important to also understand this approach's limitations.

Although there is an added complexity to the application's interaction by the end user, due to the need to learn and remember the set of gestures, this complexity is dependent on the complexity of the gestures used by the developer, and not on the specificities of the approach itself.

The trained gestures on the mobile version of the toolkit are relatively limited, as 3 gestures might not be enough to map all features from an application. Still, they allow a different type of navigation and can be used as shortcuts for some applications.

The tabletop version yielded better results on this aspect, with 6 fully trained gestures with a classification rate of over 96%. The disadvantage of this toolkit implementation is that it relies on an external microphone to capture audio and filter most of the environment noise. This type of microphones may not be available for all tabletop surfaces.

The toolkit technical evaluation also provided positive results, with low to no impacts on memory and CPU performances. However, this evaluation was executed on a relatively small application, with few trained gestures. On larger applications, and with a fully set of trained gestures available on the application, these requirements may increase.

Chapter 5

Conclusions

Applications that use acoustic sensing allied to gesture classification features to improve interaction are still a minority. This is due to the complexity on the development and implementation of these features, and a perceived lack of advantages on using them.

This situation still restricts the development of new ways of interaction and use of systems and applications.

This work was committed to create a solution to this problem. For that purpose a full approach to gesture classification through acoustic sensing was developed. This approach was developed into a toolkit that provides these features, while freeing application developers from the need to understand complex processes of machine learning or audio processing. Two versions of the toolkit were developed, one for mobile devices, and other for touch tables.

This toolkit was designed to be modular, and to allow easy integration with other applications, by exposing a well-defined interface and a simple integration process.

To validate this approach, a set of prototype applications was developed that use this toolkit and implement its features as a method of interaction (by mapping trained gestures to actions on the application), and a evaluation was executed. This evaluation focused on both the gesture classification process, to study the quality of the approach developed to classify gestures based on previous training, and on user tests to the developed prototype applications.

The results obtained from the gesture classification evaluation were very positive, with accuracy results above 90% for both the mobile devices and touch table's versions of the toolkit. The user tests also showcased the quality of the developed approach, with overall positive results and satisfaction from the 15 volunteer users.

These results corroborate the success of the developed approach, and the achievement of all objectives proposed for this work.

5.1 Future Work

Based on the results and conclusions taken from this work, there are some aspects that can be improved and extended.

Although the results demonstrated the success of the approach, there is the need for more robustness on some of its processes. Specifically, the audio processing process can be improved to allow the use of mobile applications on more diverse environments, without the risk of greatly harming the classification process. This is no easy task, as it was explained, due to the poor quality of mobile device's microphones.

The integration process could also be improved. The acquisition of data (currently executed at Android's touch listener and the approach's sound event listener) could be further simplified, to reduce the amount of coding needed for integration.

This approach does not support multi-finger gestures. To achieve this, the gesture detection and signal processing mechanisms would have to be significantly changed.

Finally, the sound intensity level classification process may also be improved, by using another alternative than the ones studied, and then integrated with the toolkit to allow another level of interaction.

Bibliography

[Bal+06] Rafael Ballagas et al. “The smart phone: a ubiquitous input device”. In: *Pervasive Computing, IEEE* 5.1 (2006), pp. 70–77.

[Bri+11] Peter Brinkmann et al. “Embedding pure data with libpd”. In: *Proceedings of the Pure Data Convention*. 2011.

[CL13] Tzren-Ru Chou and Jen-Chieh Lo. “Research on Tangible Acoustic Interface and Its Applications”. In: *Proceedings of the 2nd International Conference on Computer Science and Electronics Engineering*. Atlantis Press. 2013.

[Cod] Codehaus. *x-stream*. URL: `http://xstream.codehaus.org/` (visited on 11/29/2014).

[Dac03] LDS Dactron. “Understanding FFT windows”. In: *Application Note, www. ee. iitm. ac. in/˜ nitin/_media/ee462/fftwindows. pdf* (2003).

[Faw06] Tom Fawcett. “An introduction to ROC analysis”. In: *Pattern recognition letters* 27.8 (2006), pp. 861–874.

[Goe+14] Mayank Goel et al. “SurfaceLink: using inertial and acoustic sensing to enable multi-device interaction on a surface”. In: *Proceedings of the 32nd annual ACM conference on Human factors in computing systems*. ACM. 2014, pp. 1387–1396.

[GP14] Nicholas Gillian and Joseph A Paradiso. “The Gesture Recognition Toolkit”. In: *Journal of Machine Learning Research* 15 (2014), pp. 3483–3487.

[Hec14] Jeff Hecht. *Why Mobile Voice Quality Still Stinks—and How to Fix It*. Sept. 2014. URL: `http://spectrum.ieee.org/telecom/wireless/why-mobile-voice-quality-still-stinksand-how-to-fix-it` (visited on 10/05/2015).

[HH08] Chris Harrison and Scott E Hudson. “Scratch input: creating large, inexpensive, unpowered and mobile finger input surfaces”. In: *Proceedings of the 21st annual ACM symposium on User interface software and technology*. ACM. 2008, pp. 205–208.

[HSH11] 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*. ACM. 2011, pp. 627–636.

[KBC06] A Karlson, B Bederson, and J Contreras-Vidal. “Understanding single-handed mobile device interaction”. In: *Handbook of research on user interface design and evaluation for mobile technology* (2006), pp. 86–101.

[Li09] Yang Li. “Beyond pinch and flick: Enriching mobile gesture interaction”. In: *Computer* 42.12 (2009), pp. 0087–89.

[LJJ11] Pedro Lopes, Ricardo Jota, and Joaquim A Jorge. “Augmenting touch interaction through acoustic sensing”. In: *Proceedings of the ACM International Conference on Interactive Tabletops and Surfaces*. ACM. 2011, pp. 53–56.

[Lyo+07] Kent Lyons et al. “Gart: The gesture and activity recognition toolkit”. In: *Human-Computer Interaction. HCI Intelligent Multimodal Interaction Environments*. Springer, 2007, pp. 718–727.

[McC+12] James McCloskey et al. “Multi-Input Sensing Table: A Sensor Fusion for Drumming”. In: (2012).

[MRT12] Mehryar Mohri, Afshin Rostamizadeh, and Ameet Talwalkar. *Foundations of machine learning*. MIT press, 2012.

[Mur+08] Roderick Murray-Smith et al. “Stane: synthesized surfaces for tactile input”. In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. ACM. 2008, pp. 1299–1302.

[OST13] Makoto Ono, Buntarou Shizuki, and Jiro Tanaka. “Touch & activate: adding interactivity to existing objects using active acoustic sensing”. In: *Proceedings of the 26th annual ACM symposium on User interface software and technology*. ACM. 2013, pp. 31–40.

[P+98] Miller S Puckette, Miller S Puckette Ucsd, Theodore Apel, et al. “Real-time audio analysis tools for Pd and MSP”. In: (1998).

[Par+02] Joseph A Paradiso et al. “Passive acoustic sensing for tracking knocks atop large interactive displays”. In: *Sensors, 2002. Proceedings of IEEE*. Vol. 1. IEEE. 2002, pp. 521–527.

[PH14] Esben Warming Pedersen and Kasper Hornbæk. “Expressive touch: studying tapping force on tabletops”. In: *Proceedings of the 32nd annual ACM conference on Human factors in computing systems*. ACM. 2014, pp. 421–430.

[PKB06] Pekka Parhi, Amy K. Karlson, and Benjamin B. Bederson. “Target Size Study for One-handed Thumb Use on Small Touchscreen Devices”. In: *Proceedings of the 8th Conference on Human-computer Interaction with Mobile Devices and Services*. MobileHCI ’06. Helsinki, Finland: ACM, 2006, pp. 203–210. ISBN: 1-59593-390-5. DOI: 10.1145/1152215.1152260. URL: http://doi.acm.org/10.1145/1152215.1152260.

[Puc+96] Miller Puckette et al. “Pure Data: another integrated computer music environment”. In: *Proceedings of the Second Intercollege Computer Music Concerts* (1996), pp. 37–41.

[Pur] PureData. *AndroidLatency — PD Community Site*. URL: https://puredata.info/docs/AndroidLatency (visited on 10/05/2015).

[Qee14] Qeexo Co. *Qeexo - Makers of FingerSense Touchscreen Technology*. 2014. URL: http://www.qeexo.com/ (visited on 01/07/2015).

[RDS05] Wolfgang Rolshofen, Peter Dietz, and G Schafer. "TAI-CHI: Tangible Acoustic Interfaces for Computer-Human Interaction". In: *FORTSCHRITTE DER AKUSTIK* 31.2 (2005), p. 773.

[Sch11] Julia Schwarz. 2011. URL: http://juliaschwarz.net/publications/2011/10/02/tapsense/ (visited on 01/07/2015).

[SKN07] Beat Signer, Ueli Kurmann, and Moira C Norrie. "iGesture: a general gesture recognition framework". In: *Document Analysis and Recognition, 2007. ICDAR 2007. Ninth International Conference on*. Vol. 2. IEEE. 2007, pp. 954–958.

[SM59] Joseph C Stevens and Joel D Mack. "Scales of apparent force." In: *Journal of Experimental Psychology* 58.5 (1959), p. 405.

[Ste97] Stephen V Stehman. "Selecting and interpreting measures of thematic classification accuracy". In: *Remote sensing of Environment* 62.1 (1997), pp. 77–89.

[Wik14a] Wikipedia. *K-nearest neighbors algorithm — Wikipedia, The Free Encyclopedia*. 2014. URL: http://en.wikipedia.org/w/index.php?title=K-nearest_neighbors_algorithm&oldid=635197932 (visited on 10/05/2015).

[Wik14b] Wikipedia. *Support vector machine — Wikipedia, The Free Encyclopedia*. 2014. URL: http://en.wikipedia.org/w/index.php?title=Support_vector_machine&oldid=635600100 (visited on 10/05/2015).

Appendix A

Pure Data patch

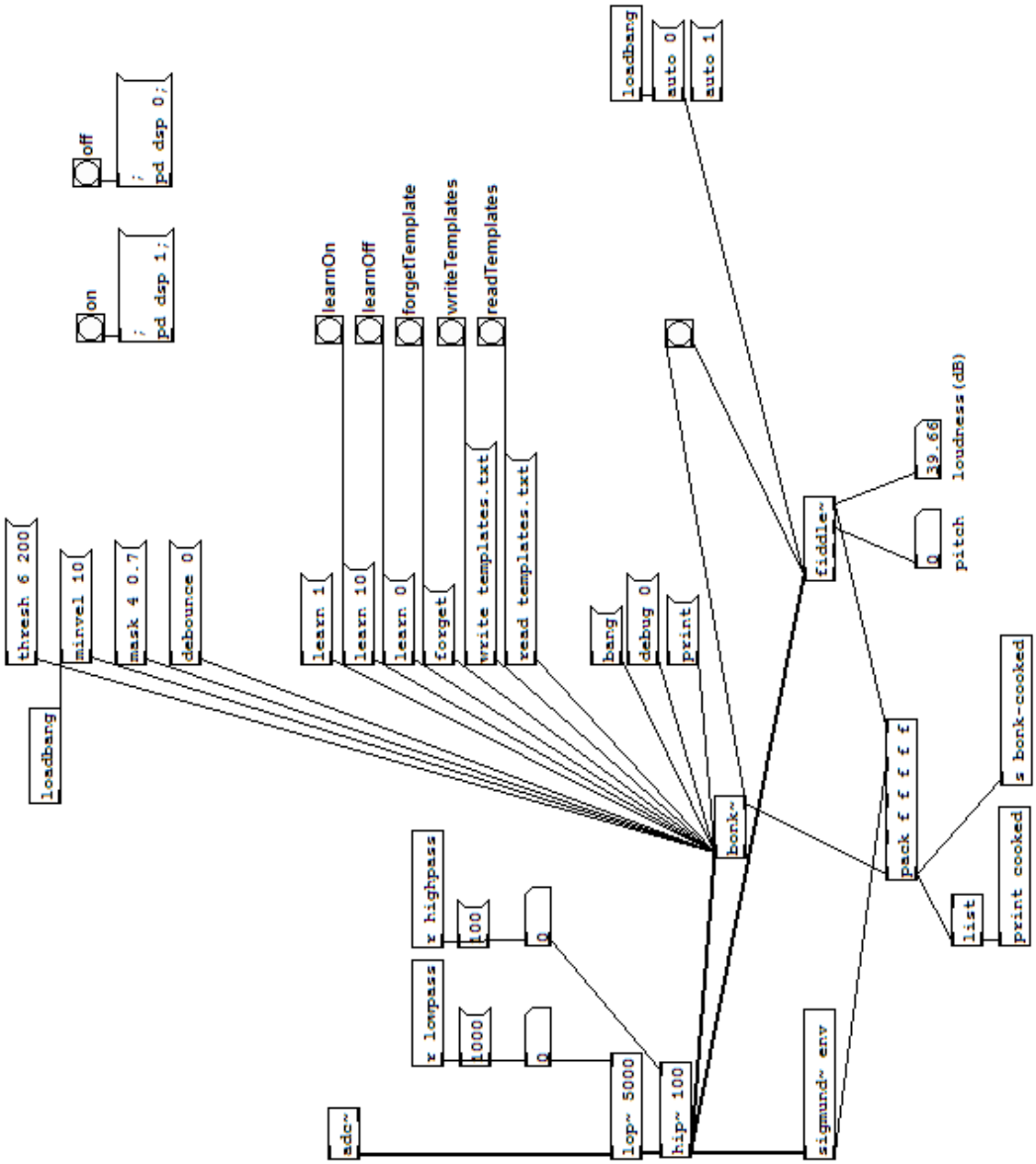


Figure A.1: Developed Pure Data patch

Appendix B

Spectrogram Application

Gesture	Frequencies	
	Fundamental	**Partials**
TAP	200Hz	400Hz
NAIL	200Hz	300Hz, 400Hz
KNOCK	200Hz	400Hz, 1100Hz, 1300Hz

Table B.1: Spectrogram application results

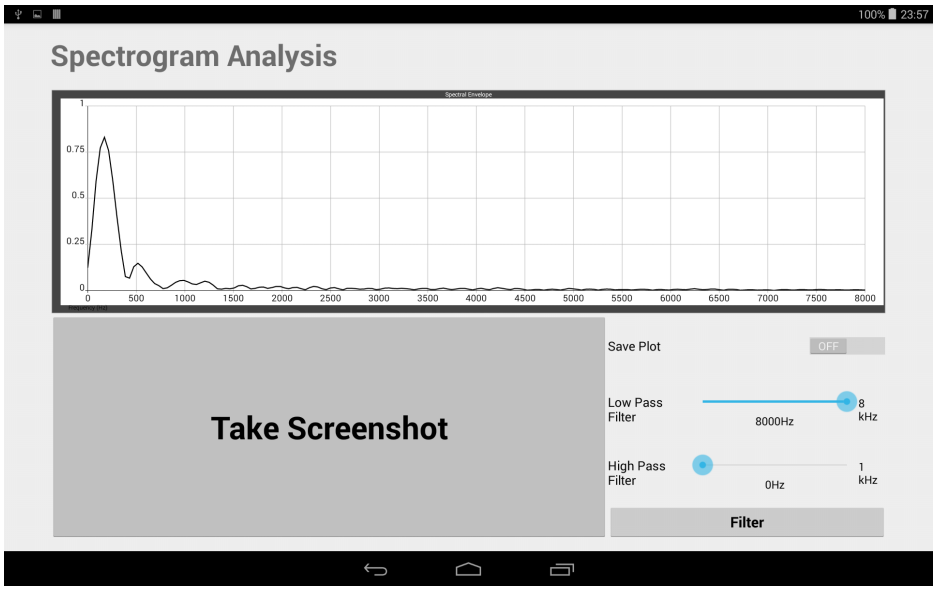


Figure B.1: Spectrogram application results for the TAP gesture

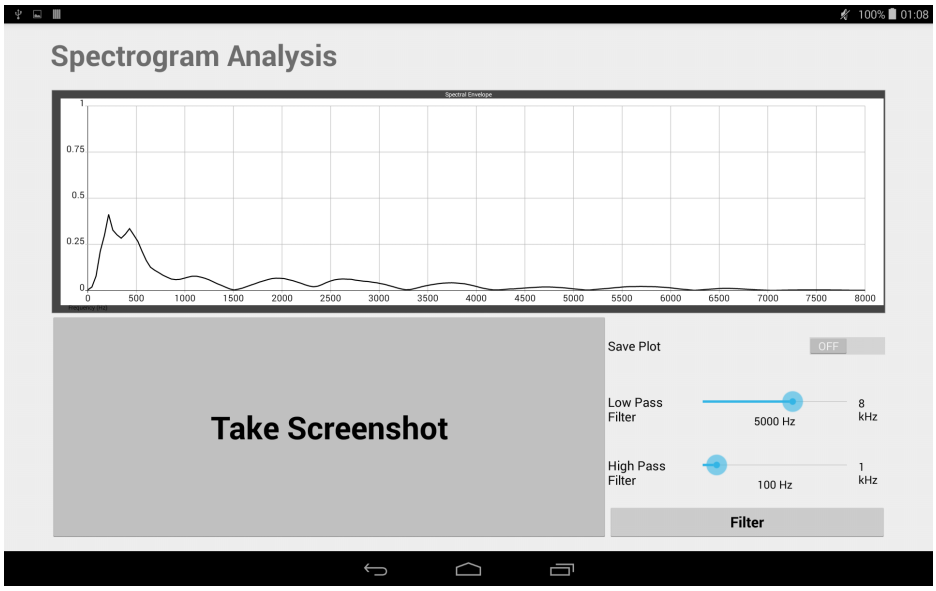


Figure B.2: Spectrogram application results for the NAIL gesture

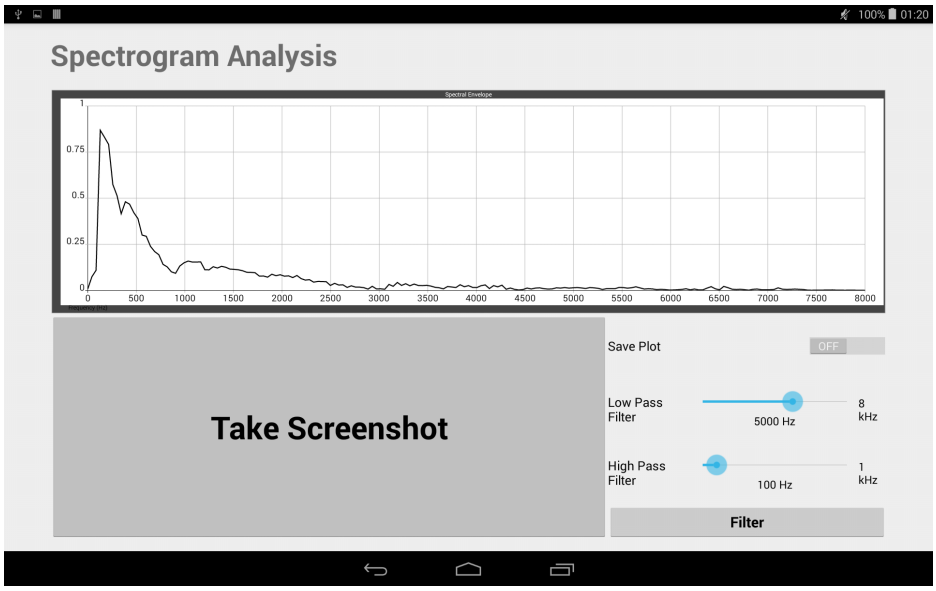


Figure B.3: Spectrogram application results for the KNOCK gesture

Appendix C

Mobile Toolkit's Tables of Confusion

		Prediction outcome		
		p	**n**	**Total**
Actual value	**p′**	20	0	P′
	n′	6	54	N′
	Total	P	N	

Table C.1: Table of confusion for the TAP gesture

		Prediction outcome		
		p	**n**	**Total**
Actual value	**p′**	18	2	P′
	n′	1	59	N′
	Total	P	N	

Table C.2: Table of confusion for the NAIL gesture

		Prediction outcome		
		p	**n**	**Total**
Actual value	**p′**	17	3	P′
	n′	4	56	N′
	Total	P	N	

Table C.3: Table of confusion for the KNOCk gesture

		Prediction outcome		
		p	**n**	**Total**
Actual value	**p′**	11	9	P′
	n′	3	57	N′
	Total	P	N	

Table C.4: Table of confusion for the STYLUS gesture

Appendix D

User Tests Guide

This document presents the protocol specified for the user tests to the ImpactPaint application, developed to study the integration of our gesture classification toolkit.

This tests are going to be executed with 15 users. All testing sessions will be conducted on a calm, noise-free environment.

Tests will be divided in two parts. First this document will be delivered and some brief explanation will be given about the application. Then, we will request the execution of the three tasks, described below.

Information collected

For each task, the following information will be collected: Time of execution Number of errors Number of actions

Gestures mapping

The next table describes the features provided by each gesture. These gestures will be used on the following tasks.

Name	Feature
TAP	Simple drawing, with a black colored pencil
NAIL	Eraser
KNOCK	Circle stamp (toggle circle drawing)

Tasks

1. Draw a ”#” symbol on the screen;
2. With a nail tap activate the eraser, erase one of the lines, and draw another line;
3. Knock on the screen to activate the circle brush and draw 2 circles, knock again to deactivate the circle brush, activate the eraser and erase one of them, and draw a straight line over the remaining circle.

Appendix E

Tabletop Toolkit's Tables of Confusion

		Prediction outcome		
		p	**n**	**Total**
Actual value	**p′**	20	0	P′
	n′	4	96	N′
	Total	P	N	

Table E.1: Table of confusion for the TAP gesture

		Prediction outcome		
		p	**n**	**Total**
Actual value	**p′**	18	2	P′
	n′	6	94	N′
	Total	P	N	

Table E.2: Table of confusion for the KNOCK gesture

		Prediction outcome		
		p	**n**	**Total**
Actual value	**p′**	16	4	P′
	n′	1	99	N′
	Total	P	N	

Table E.3: Table of confusion for the NAIL gesture

		Prediction outcome		
		p	**n**	**Total**
Actual value	**p′**	14	6	P′
	n′	2	98	N′
	Total	P	N	

Table E.4: Table of confusion for the PUNCH gesture

		Prediction outcome		
		p	**n**	**Total**
Actual value	**p′**	17	3	P′
	n′	2	98	N′
	Total	P	N	

Table E.5: Table of confusion for the SLAP gesture

		Prediction outcome		
		p	**n**	**Total**
Actual value	**p′**	20	0	P′
	n′	0	100	N′
	Total	P	N	

Table E.6: Table of confusion for the PEN gesture

Appendix F

UML Classes Diagram

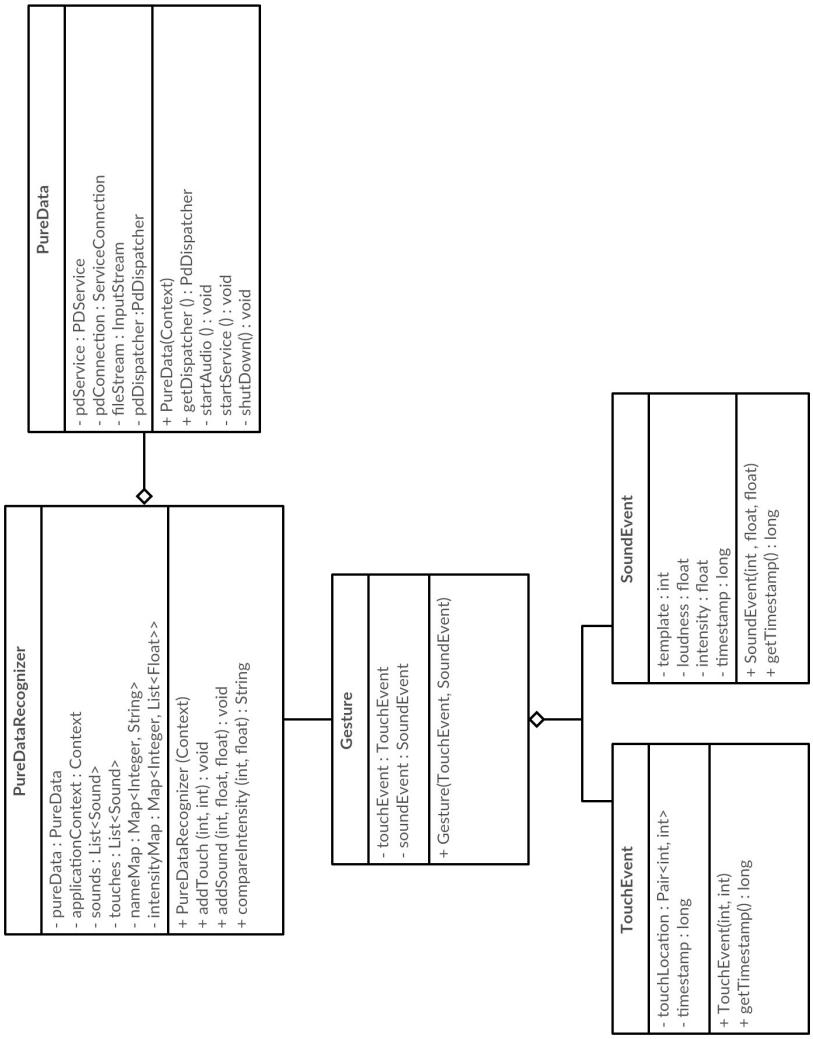


Figure F.1: Mobile device's toolkit Classes diagram