Gestural Music Expression Application using smartphone's embedded sensors in Android Operating System

By

Luis Joglar

Project Report 2: Discussion and Evaluation

A report on project work carried out for the degree of

BSc (Hons) Music Technology and Audio Systems

1st Supervisor Dr Matthew Stephenson

2nd Supervisor Mr Braham Hughes

12/05/17

Abstract

Gestural interaction for musical expression is of increasing interest within the music technology community, as shown by conferences like NIME. This paper describes the elements used to develop a gestural music expression interface by using day-to-day and machine learning technologies. Discussion on improvements and outlining of limitations and problems found in the process are also carried out.

List of Contents

Abstract 2
List of Contents 3
List of Figures 4
Introduction 5
Chapter 1: Design and development 5
1.1. Machine learning algorithms 5
1.1.1. Supervised vs. Unsupervised 6
1.1.2. Dynamic Time Warping (DTW) 6
1.2. Technical Aspects 7
1.2.1. Android 7
1.2.2. Integration of GRT (Old NDK vs. New NDK) 7
1.2.3. Accelerometer Data (SDK vs. NDK) 9
1.2.4. GRT: Training the pipeline and real time recognition 10
1.2.5. MIDI vs OSC 12
1.3. Architecture and Design 12
1.3.1. High level structure 13
1.3.2. Application Flow 13
1.3.3. Low level structure 15
MainActivity 15
Main2Activity 16
Native library 16
Chapter 2: Evaluation and results 16
2.1. Machine learning algorithm and GRT 17
2.2. Sensors 17
2.3. Sample Frequency 18
2.4. User Interface 18
2.5. Gestures 18
2.5.1. Gesture 1 19
2.5.2. Gesture 2 21
2.5.3. Gesture 3 22
2.6. Latency 23
2.7. Testing 24
Chapter 3: Conclusions and further work 24
References 26
Appendices 27
Appendix 1 27
Appendix 2 28

List of Figures

Figure 1. General flow diagram of the applicarion 13

Figure 2. Flow diagram of the training process 14

Figure 3. Flow diagram of the real time recognition process........ 15

Figure 4. User Interface for the application 18

Figure 5. Representation of the axes of the accelerometer in a smartphone 1919

Figure 6. Diagram of the movement for gesture 1 200

Figure 7. Graphs of the data from the three axes of the accelerometer for gesture 1 200

Figure 8. Diagram of the movement for gesture 2 211

Figure 9. Graphs of the data from the three axes of the accelerometer for gesture 2 222

Figure 10. Diagram of the movement for gesture 3 222

Figure 11. Graphs of the data from the three axes of the accelerometer for gesture 3 233

Introduction

The evolution of computing science, since Blaise Pascal created the first mechanical calculator in the 17th century, has lead to an increase of interest in areas like the machine learning and the human-computer interaction (Helander, Landauer, & Prabhu, 1997). These two areas build attention in the music technology community due the new capabilities these can introduce, especially in the field of EDM.

Due to the fact that EDM music is mainly performed on a computer, expressivity during live performances is usually poor. Studies suggest that the public is getting used to this, an improvement in the expressivity of performances would increase the interest of the audience (Mitchell & Heap, 2011).

This project aims to create a gesture based music expression interface based on day-to-day technologies. To do so, an Android application will be developed using the information from the embedded sensors. The system will make use of machine learning algorithms to allow users to train the application to recognize their own gestures. Once the gestures are recognized, wireless communication with a computer will be established for interaction with music software.

Chapter 1: Design and development

1.1. Machine learning algorithms

This project uses a machine learning external library for the training and recognition of the gestures, called Gesture Recognition Toolkit (GRT). This is an in-depth library offering a wide range of algorithms. A good understanding of these algorithms and their differences is important for choosing the appropriate for each circumstance.

1.1.1. Supervised vs. Unsupervised

The two main algorithm groups for handling real-time patterns recognition are: supervised and unsupervised.

Supervised learning refers to algorithms in which the training is done through a set of data values where the solution is already known. These algorithms rely on the idea that there is a relationship between the data and the result (Tucker, 2004).

Two main types of supervised learning algorithms are defined as regression algorithms, in which the predicted result is a value within a continuous function, and classification, algorithms in which the solution is a discrete value from a set of categories.

Unsupervised learning allows the system to predict or classify without previous knowledge of the results. These algorithms deduce categories by clustering the data or using other methodologies like Neural Networks.

The main goal of this project is to train the Android application to recognize gestures created by the user. Users will record gestures and after the training, the system will recognize the gestures in real-time. That means the training data will contain the solution information and the output is a discrete value, therefore the types of algorithm of interest for this project are: supervised learning classification algorithms.

1.1.2. Dynamic Time Warping (DTW)

For gesture recognition, especially in musical applications, classification algorithms such as Hidden Markov Models and Artificial Neural Networks are the most commonly used (Joselli & Clua, 2009).

Another, not as common but very effective, algorithm is the DTW. The main characteristic of DTW is the capability to ignore differences within the time domain. DTW is highly effective when the analysed data is a time-oriented function, and the differences in speed are a factor not to consider.

An example of this would be the tempo. The training gestures are probably not going to be recorded at any specific tempo, but during the performances the user will, more likely,

move at the tempo of the song. For this reason DTW is the best algorithm choice for the project.

1.2. Technical Aspects

1.2.1. Android

One of the main project aims was the use of every day technology instead of creating a specific device, for this reason it has been developed as an Android application. Android is the most extended operating system for smartphones. This makes it the best candidate for this development.

A Java Software Development Kit (SDK) for Android is provided for free by Google. This SDK is equipped with the necessary tools for Android development in the form of a series of Application Programming Interfaces (APIs) to easily access and utilise all of the smartphone's features and capabilities. A Java Integrated Development Environment (IDE) called Android Studio is also offered for free. This provides full access to the Android tools, a system to compile into the target device and also a device emulator and debugging tools.

Android also allows developing applications, or part of them, in C/C++. A package called Native Development Kit (NDK), which includes a series of C++ APIs to access the smartphone capabilities, is also provided. These APIs are more limited than the Java ones, and are also more complex to use.

1.2.2. Integration of GRT (Old NDK vs. New NDK)

One of the biggest challenges during the development of this project was the integration of the GRT C++ library for gesture recognition into Android.

The first step was the creation of an Android application with integrated C++ (native) libraries. Following instructions from the developer android website, ("Add C and C++ Code to Your Project | Android Studio," n.d.) was fairly mild to add simple existing libraries or to

create a new one. These steps allowed the creation of a native library where new C++ code could be applied for use within this application.

The first obstacle in the development process of this application was the inclusion of a complex C++ library by making it a native library. In order to learn what the procedure was, books and online resources were researched. All the information found was not up to date due to the continuous development of the Android operating system and the Android Development Studio.

A decision was to be made, either use an older version of the Android environment where instructions were commonly available, or to use the newest version attempting to recreate the steps from the older version into the new system.

It is arguable that the use of the latest stable development environment is recommendable. For that reason the second option was pursued. Two online resources where especially useful:

- The first one is a tutorial using the old integration methodology to integrate the GRT in Android. ("Compilation du Gesture Recognition Toolkit sur Android," n.d.)
- The second one is an example of integrating a complex library into the new system. ("manimaul/AndroidNativeLibExample," n.d.)

By reproducing some of these steps into the latest system, following the new methodologies, and after several days of trial and error, the integration was accomplished.

This process can be summarised in two steps:

1. Portability of the library into the Java environment. This step allows the Java environment to understand the library. A file containing all the information needed by the system was created using a program called SWIG. This program uses as an input a file determining the address of all the files from the library needed. This file can be found in Appendix 1. The command line used to execute SWIG is as follows:

```
swig -c++ -java -package com.example.ljoglar.swig -outdir
../java/com/example/ljoglar/swig/ -o grt_wrap.cpp grt.i
```

Being grt_wrap.cpp the output file and grt.i the input file.

2. Compilation of the C++ library. The addresses of all the files of the library together with the new file grt_wrap.cpp need to be added into the CMakeList.txt, so Android Studio can compile the library.

   The list of addresses needs to be added under the following command:

```
add_library(
          # Sets the name of the library. grt-lib
          # Sets the library as a shared library. SHARED

          # Address list
)
```

   And at the end of the file its name has to be added into:

```
target_link_libraries(
          # Specifies the target library.
                    native-lib
                    grt-lib
                    android
                    app-glue
                    EGL
                    GLESv1_CM
          # Links the target library to the log library
          # included in the NDK.
          ${log-lib} )
```

1.2.3. Accelerometer Data (SDK vs. NDK)

One of the most important things in this project is access to the information that allows the application to identify the gestures made by the user. The sensors embedded in the smartphone provide this data. This project solely uses the accelerometer sensor, which provides information of the proper acceleration of the device in three axes. Proper acceleration refers to the acceleration of an object relative to a free fall (Taylor & Wheeler, 1992). If an object is in free fall, the proper acceleration is 0, if an object is at rest, the acceleration in the vertical axis will be equal to the gravitational acceleration 9.81m/s^2. This information is not very intuitive to interpret, but it can be understood as the acceleration of the device with respect to the user.

There are two ways to access the sensor data from the Android device: through the SDK API, or accessing it via NDK in the native library. As the data will be used in the native side of the program, the logical way to access this information is in the NDK. This procedure was attempted but due to the complexity of the native API, and the limited time to develop the project, the easier option was finally implemented that being the SDK access to the sensors.

Once the Java side of the program reads the data from the three axes of the accelerometer, it is sent to the native side to be stored and utilized.

1.2.4. GRT: Training the pipeline and real time recognition

The GRT recognition system works around a class called "Pipeline"; this class provides the elements required to create a system that can learn to recognize gestures. The library also implements a class for each of the classification, regression and clustering algorithms that can be used.

As this application uses the classification algorithm DTW an instance of it needs to be created in order for the pipeline to use its methods.

```
GestureRecognitionPipeline pipeline;
DTW dtw;
pipeline.setClassifier(dtw);
```

Once the class for the algorithm is instantiated and set into the pipeline as the classifier algorithm, the training process can start. The pipeline object has the main functionalities needed to train the system and to recognize the gestures in real time.

The next step is to create the data structure that is to be fed into the pipeline. The DTW needs the data from the sensors to be stored in a GRT data type called: TimeSeriesClassificationData ("Time Series Classification Data — NickGillianWiki," n.d.). This data structure is used for supervised temporal based learning problems. This structure is constituted by an N-dimension time series of length M.

In this case the time series is an array of three dimensions, for the three axes of the accelerometer. The length will be determined by the sample frequency of the sensors and the duration of the gesture. This array of data is stored in a GRT type called MatrixFloat, which is a multidimensional array of floats.

Every time a gesture is recorded, a new MatrixFloat is added into the TimeSeriesClassificationData with an identifier of the class label it belongs to. The class label identifies each gesture.

The following code shows the initialization of the variables needed for the training and the main actions described.

```
int dimensions = 3;
TimeSeriesClassificationData trainingData;
trainingData.setNumDimensions(dimensions);
MatrixFloat trainingSample;
UINT maxClassLabel;
VectorFloat sample(dim);

trainingSample.push_back(sample);

trainingData.addSample(maxClassLabel, trainingSample);
trainingSample.clear();
```

Finally, once the TimeSeriesClassificationData has all necessary sets of data to train the system, the pipeline can be tought by calling its train method.

```
pipeline.train(trainingData);
```

This function returns a boolean, allowing to check if the training was successful. Once the training is effective the real time recognition can start. The pipeline object offers two ways of recognition, both using the function Predict: one where a simple sample is passed in and a second one where the data is sent in blocks of samples. As the goal for this project is to use it in real time to interact musically with the performance, the latency of the system has

to be as small as possible. For that reason this application uses the first option, sending the samples into the Predict function every time the sensors is read.

The Predict function returns true if a prediction has been made, false otherwise. When a prediction is made the class label of the recognized gesture is stored into the pipeline, and can be accessed by the method getPredictedClassLabel.

```
pipeline.predict(sample);
pipeline.getPredictedClassLabel();
```

1.2.5. MIDI vs. OSC

The last stage of the application is the communication with the computer the performer will interact with. Android SDK provides MIDI communication protocol integration within the APIs, but this system works through a USB cable. To be able to communicate wirelessly, another Android API, to create a TCP/IP connection, is needed.

Another option was the use of OSC instead of MIDI. A library called JavaOSC offers the possibility to communicate wirelessly to a target device connected in the same Wi-Fi network as the origin device. The library manages the TCP/IP connection just by knowing the IP address of the target in the network and the port the target device will listen to.

As the JavaOSC facilitates wireless connection and the OSC protocol allows more flexibility of communication than MIDI, it was the implemented option.

1.3. Architecture and Design

In this section the main structure of the code will be described, alongside how the communication between the two development environments works. The flow of the application and the user interaction is also detailed.

1.3.1. High level structure

As explained previously, this application is developed using two environments in continuous communication. The following figure shows the general flow of the application, and the communication between the NDK (or native libraries written in C++) and the SDK side of the program developed in Java.

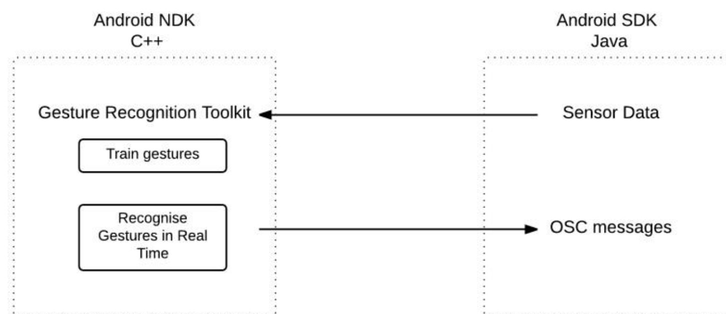

Figure 1. General flow diagram of the applicarion

The data from the sensors is accessed from the Java side using the sensor's API and is sent as floats into the native side where they are stored. This data is then and used to train the system, or used in real time to recognize the gestures already trained. Once a gesture is recognized a variable identifying the gesture is sent to the Java side that uses an external library to send an OSC message to a device to interact with it.

1.3.2. Application Flow

The following figure illustrates the flow of the application during the training phase. The user can start (and stop) recording data for each gesture. If the user dictates that the recorded data is not representative enough of the target gesture, they can discard the last trial before attempting again. Once the user has recorded the current gesture a few times, they can select the option to record a new gesture and repeat the process for each gesture they want to use. Once all the gestures are recorded, by pressing the train button, the system is trained with all the recorded data.

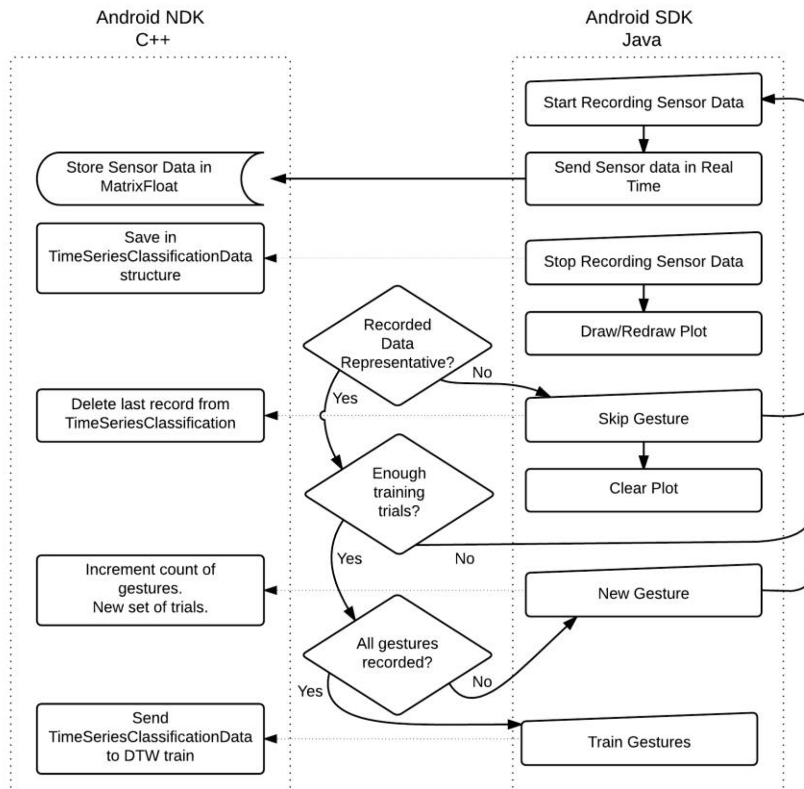

Figure 2. Flow diagram of the training process

Once the application is trained the user can press the button Gesture It, to create a connection via TCP/IP protocol with the computer. To create this connection the two devices must be connected to the same Wi-Fi network. The IP of the computer must be determined in the settings of the application, along with the port the computer will listen to, to receive the messages. Once the connection is established the application will send an OSC message every time a gesture is identified.

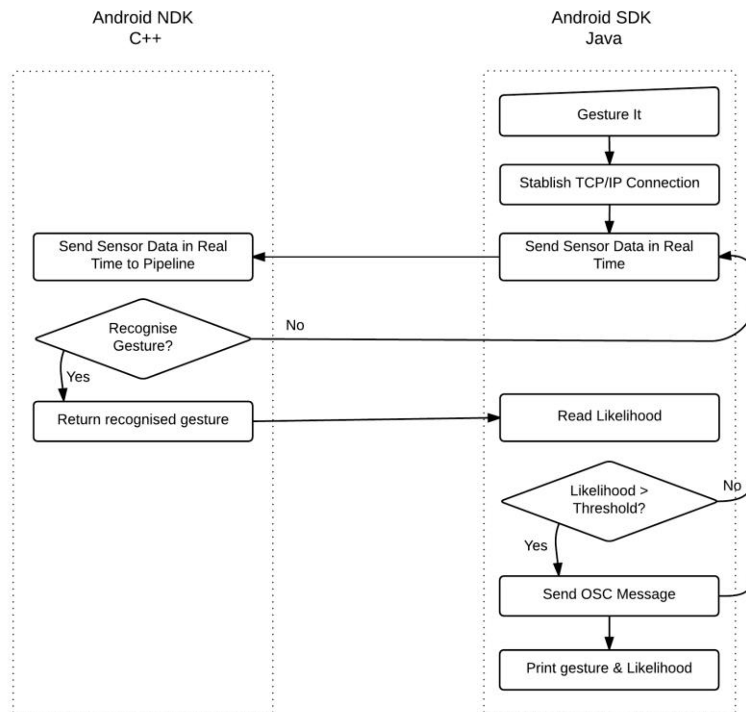

Figure 3. Flow diagram of the real time recognition process

1.3.3. Low level structure

The code of the application is divided in three main files, each of them containing a class. In Android every window the application has needs a class to control all the graphical elements. The logic of the program is implemented as methods of these classes. The third file is the native library created to interact between the GRT library and the Java side of the Android application.

In the Appendix 2 a class diagram of each of these classes can be found.

MainActivity

In the MainActivity class, all the logic for the buttons the user can interact with can be found. It also implements the methods used to read the information from the accelerometer and to send the data to the native library, as well as to create the TCP/IP connection and to create and send the OSC messages. The graph view where the graphs show the accelerometer data is also implemented and controlled from this class.

Main2Activity

This class controls the graphical elements within the settings window. This information is sent to the native library where all of these values are saved and used to set parameters of the GRT.

Native library

In the native library the Gesture class is implemented. This class stores the instances of the GRT's pipeline and classification algorithm used within the application. In this file several functions controlling the communication between the Java classes and the native library are implemented as well. These functions allow bi-directional communication with the Java classes. For this communication to work, these functions need to be declared in the Java class as well.

The Gesture class also implements several methods that act as a middleman between the GRT pipeline and the functions that manage the communication with the Java side.

Chapter 2: Evaluation and results

The resulting application is considered to be a prototype rather than a final application. This means that the current state is not reliable enough to work under all circumstances. The application needs further testing and analysis of the characteristics that make gesture recognition reliable, to improve its consistency. That being said, the application works and can be used after some study and optimization; the user must try different gestures and settings to find the best calibration and movements.

The following points will discuss and critically analyze the different aspects of the application that can be assessed to continue its development. Also some tests from a user perspective will be described and analyzed.

These tests were carried out as part of the development process, rather than as a specific process itself. Their purpose was to find out if features in development worked as it was supposed to, or if the whole system was working properly after a new feature.

Further testing on alternative development decisions to empirically resolve the best option will also be discussed.

2.1. Machine learning algorithm and GRT

DTW is used as this algorithm is designed for finding patterns in data over time ignoring the differences in speed. The resulting recognition ratio is acceptable. Other real-time recognition algorithms like Hidden Markov Models and Gaussian Mixture Models are also implemented in the GRT. The author considered it would be worthwhile testing which one offers a higher ratio of recognition.

As explained previously in section 1.2.4, the Predict function on the Pipeline object accepts two methods of inputting data. For this application the data is sent into the pipeline every time a new sample is read from the sensors. This was decided to obtain the fastest reaction to the gestures made by the user. It has been observed that the gestures are recognized before finishing the movement, providing a good reaction speed.

A way to test the accuracy of the second option of the predict method without sacrificing reaction speed could be the creation of a circular buffer where the new sample was added every time. Then using this buffer for the prediction.

It would be also interesting to learn more about how the predict function works, this would provide more options for improving the design.

It is also considered important to continue studying the GRT to facilitate greater use of the library. The actual program makes a fairly basic use of the capabilities the GRT offers.

2.2. Sensors

This application uses the accelerometer installed within the smartphone. A better understanding of the different sensors the smartphones have embedded would also provide further methods to receive more data for gesture identification.

2.3. Sample Frequency

The sample frequency is also a parameter that could be interesting to test. For this application a preliminary test was carried out along with some research, and it was found that a sample frequency of 33Hz provided enough information to recognize the gestures.

Further testing on varying the sample frequency to find an optimal value without sacrificing latency time could be very useful to improve the accuracy of the recognition.

2.4. User Interface

The user interface was designed to respond to basic user interaction needs. A button-based design was implemented to allow the control needed for the training process and to start the real-time recognition. Some text views were added to provide some feedback.

During the development the graph view was added to the interface. This provided the user with better feedback of the gestures made during the training phase. Thanks to this feedback, the feature to skip the last gesture recorded was introduced. This responds to the fact that during the recording mistakes can happen. This allows the user to discard that recording.

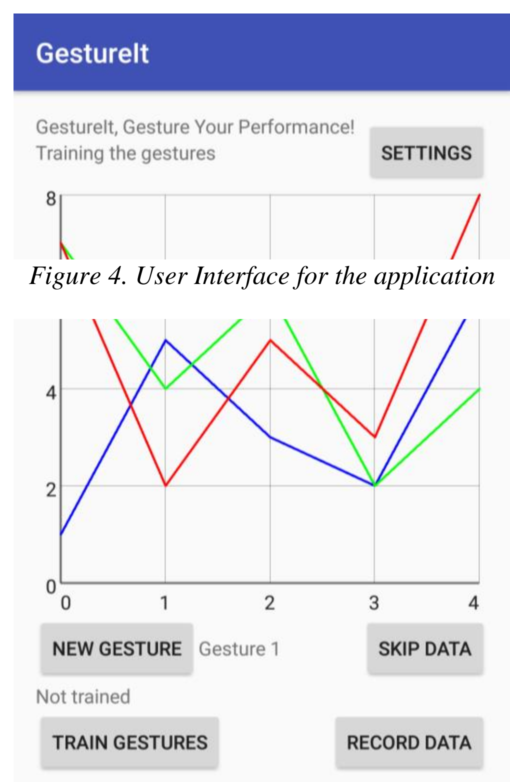

Figure 4. User Interface for the application

Whilst the design of the interface responded to the needs during the development process, the small buttons clearly limit the action of the user during the training process. This is due to requiring a very specific action to start and stop recording the data. An interaction based on touch gestures made in a bigger area of interaction instead of buttons, would provide more freedom to the user.

2.5. Gestures

The gestures made by the user are of great importance because the application doesn't react equally to every gesture. For that reason the user must

test the application with different gestures in order to find the gestures that work best for them.

Below an analysis of three gestures, with different resulting reliability, is carried out. The percentage of reliability has been determined by training the application with one gesture and replicating the gesture 25 times, measuring the amount of times the application recognizes it. These tests have been carried out by the author, it is arguable that the same gestures made by another user will get similar but different results.

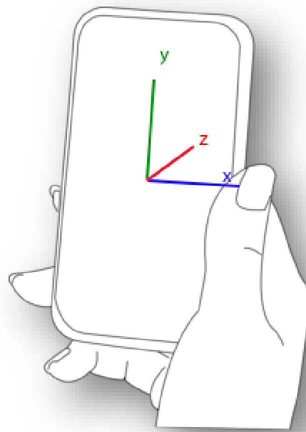

Figure 5. Representation of the axes of the accelerometer in a smartphone

2.5.1. Gesture 1

This gesture is made by holding the phone as shown in fig. 5 and rotating it around the X axis while moving the hand down and returning to the original position.

Figure 6. Diagram of the movement for gesture 1

The following figure shows the accelerometer data from the three axes for a series of six trials doing this gesture. This data is stored and used to train the GRT pipeline.

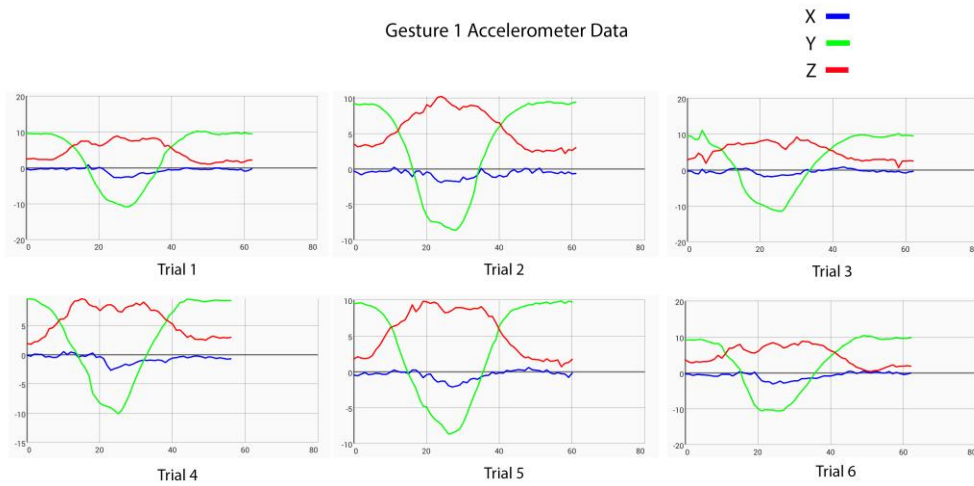

Figure 7. Graphs of the data from the three axes of the accelerometer for gesture 1

It can be seen how the data of the X axis is quite stable around 0 as the Y shows how there is a deceleration followed by an acceleration, while the Z does the inverse.

This gesture is proven to be very reliable through out the tests. The graphs show a clear pattern with small differences. It is an easy move to make, and the ratio of recognition is very high, at 92%.

2.5.2. Gesture 2

The second gesture is made by holding the phone as shown in the figure 5 without any rotation whilst lifting the hand and returning to the start position as per fig. 8.

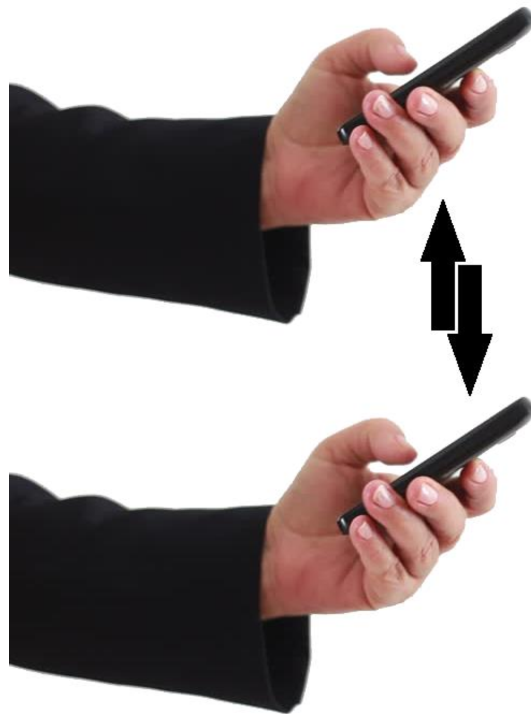

Figure 8. Diagram of the movement for gesture 2

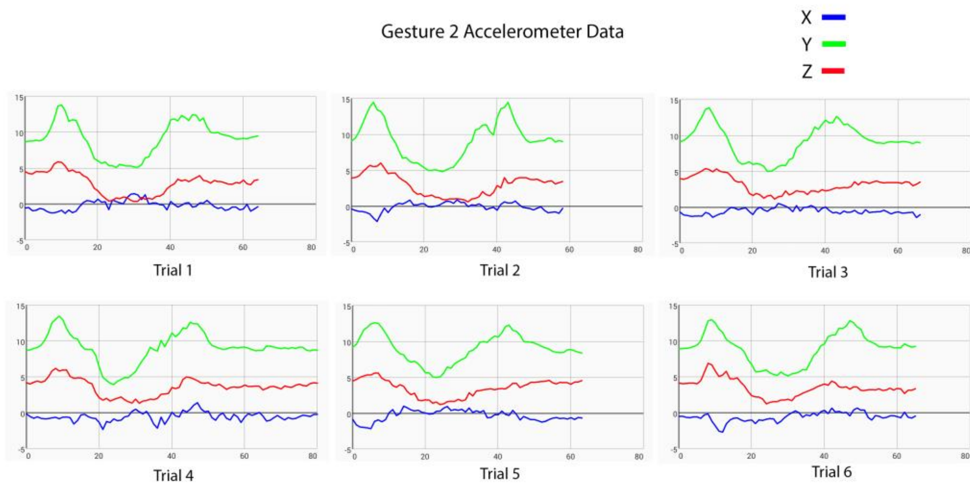

Figure 9. Graphs of the data from the three axes of the accelerometer for gesture 2

As shown in fig. 9, the X axis oscillates irregularly around 0, making it more unstable than in the first gesture. The Y and Z axes follow the same pattern in this case, where they have two peaks in acceleration. The Y axis has more pronounced variations. This second gesture is also quite reliable, but not as much as the first one with a percentage of 80%. It can be seen how the general pattern in this case is less consistent than in the first example.

2.5.3. Gesture 3

The last gesture is made by holding the phone in the same position as the previous gestures and rotating the phone on the Y axis. This is performed while exercising an external rotation of the shoulder and coming back to the original position.

Figure 10. Diagram of the movement for gesture 3

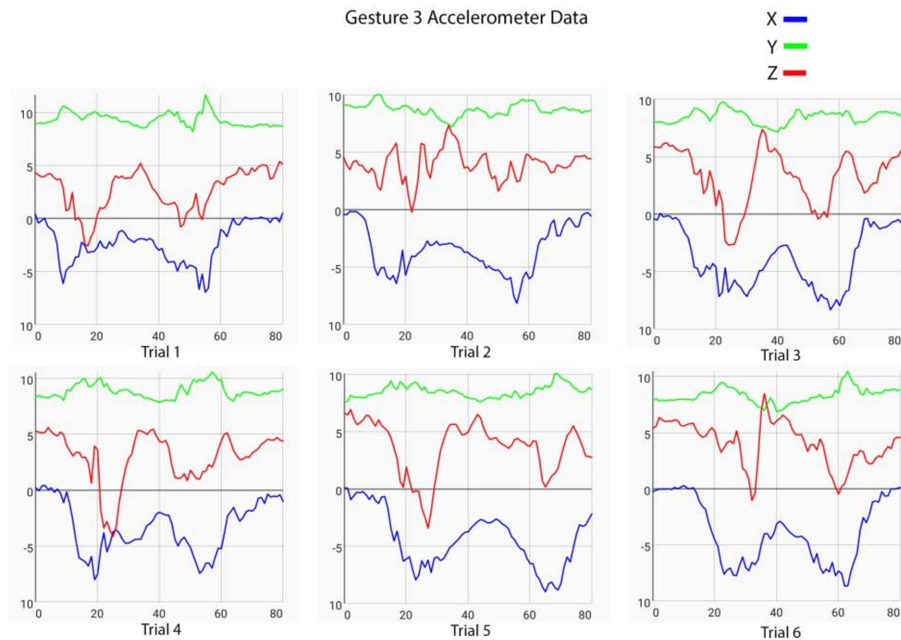

Figure 11. Graphs of the data from the three axes of the accelerometer for gesture 3

This last gesture is more complex. It is observable in the graph how in this case the Y axis is the one with the least acceleration change (due to it being the rotating axis), but it is not firm. The X axis has a recognizable but unsteady pattern, while the Z axis is the more unstable of the three. This last gesture is the least reliable of the three with a ratio of 52%.

Similar tests to the previous have been carried out to analyse the flexibility of the recognition system. The ability to discard false positives when similar gestures are made has been identified. When a reliable gesture has been trained, the system is sufficiently flexible to recognize it with little differences, but discards similar gestures. For example doing the same movement but holding the phone in a different angle.

2.6. Latency

As discussed previously the design of the program has been made with the goal of having the smallest latency possible. It has been detected when more than six gestures are trained into the application, latency starts to increase.

Deeper studies on how the predict method in the Pipeline class works could allow finding solutions to diminish the latency problem. It would be interesting to test the results both in reliability, and in time, when using the second option of the Predict method, where a time series is passed instead of a single sample.

2.7. Testing

Through out the development of the application several test phases were carried out. Thanks to these, improvements on the algorithm have been implemented.

- Likelihood filter. This is a filter to avoid false positives. After a gesture has been recognized the GRT assigns a percentage of likelihood for the movement made in relation to the data used for the training. The threshold for this filter can be modified on the fly in the settings window.
- Trimming of the recorded data. The GRT provides a method to trim the recorded data from the sensors at the beginning and the end of the array. Two parameters can be set to adjust this option: a normalized threshold, and a maximum percentage. These two parameters where implemented as options in the settings window.
- Timeout. The amount of milliseconds that the application will wait after a gesture has been recognized to continue recognizing. This parameter can be specified in the settings window.

Chapter 3: Conclusions and further work

This project aimed to develop a gestural music expression interface with day-to-day technologies. An Android application using the data from the embedded sensors in the smartphone to recognize gestures has been implemented. Machine learning technology is utilised to allow the user to train the gestures to be recognized within the application. Once

the system recognizes the gestures in real time (in order to musically interact with a device such a computer) a TCP/IP connection is created to send OSC messages in real time.

The GRT, an in-depth library for gesture recognition using machine learning algorithms developed in C++, has been fully integrated into the Android environment.

Whilst the main goal has been accomplished and the application works, improvements on the accuracy and reliability of the recognition system need to be carried out in order to develop an application that can be used by the general public. Through out the report, ideas on further study and alternatives to the current algorithm and design decisions have been proposed and discussed. Importantly, further study on the characteristics that make some gestures work better would provide useful information to improve the system making it a very interesting tool for musicians and performers.

There are some features that would improve the application in order to be considered a finalised product:

- System to save settings and gestures.
- Possibility to send continuous control messages when not recognizing gestures.
- Option to stop and resume recognition and OSC message communication.

References

Add C and C++ Code to Your Project | Android Studio. (n.d.). Retrieved May 12, 2017, from https://developer.android.com/studio/projects/add-native-code.html#new-project

Compilation du Gesture Recognition Toolkit sur Android. (n.d.). Retrieved May 12, 2017, from http://www.code-flakes.com/2015/12/compilation-du-gesture-recognition.html

Helander, M., Landauer, T. K., & Prabhu, P. V. (1997). *Handbook of human-computer interaction*. Elsevier.

Joselli, M., & Clua, E. (2009). grmobile: A framework for touch and accelerometer gesture recognition for mobile games. In *2009 VIII Brazilian Symposium on Games and Digital Entertainment* (pp. 141–150). IEEE. Retrieved from http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=5479100

manimaul/AndroidNativeLibExample. (n.d.). Retrieved May 12, 2017, from https://github.com/manimaul/AndroidNativeLibExample

Mitchell, T. J., & Heap, I. (2011). SoundGrasp: A gestural interface for the performance of live music. Presented at the Proceedings of the International Conference on New Interfaces for Musical Expression, 30 May - 1 June 2011, Oslo, Norway,Proceedings of the International Conference on New Interfaces for Musical Expression, 30 May - 1 June 2011, Oslo, Norway, Oslo, Norway. Retrieved from http://www.nime2011.org/proceedings/papers/M13-Mitchell.pdf

Taylor, E. F., & Wheeler, J. A. (1992). *Spacetime Physics: Introduction to Special Relativity* (1992 edition). New York, NY: W. H. Freeman.

Time Series Classification Data — NickGillianWiki. (n.d.). Retrieved May 9, 2017, from http://www.nickgillian.com/wiki/pmwiki.php/GRT/TimeSeriesClassificationData

Tucker, A. B. (Ed.). (2004). *Computer Science Handbook, Second Edition* (2 edition). Boca Raton, Fla: Chapman and Hall/CRC.

Appendices

Appendix 1

File grt.i

```
%module grt

%{
#include "GRT.h"
using namespace GRT;
%}

%inline %{
typedef unsigned int UINT;
%}

%rename(assign) GRT::IndexDist::operator=;
%rename(assign) GRT::DTW::operator=;

%include "Util/GRTTypedefs.h"
%include "Util/GRTException.h"
%include "Util/Log.h"
%include "Util/ErrorLog.h"
%include "Util/DebugLog.h"
%include "Util/WarningLog.h"
%include "Util/MinMax.h"
%include "Util/FileParser.h"

%rename(assign) GRT::TimeSeriesClassificationSample::operator=;
%rename(get) GRT::TimeSeriesClassificationSample::operator[];
%include "DataStructures/TimeSeriesClassificationSample.h"

%rename(assign) GRT::TimeSeriesClassificationData::operator=;
%rename(get) GRT::TimeSeriesClassificationData::operator[];
%include "DataStructures/TimeSeriesClassificationData.h"

%include "DataStructures/MatrixFloat.h"
%include "DataStructures/Matrix.h"
%include "DataStructures/Vector.h"
%include "DataStructures/VectorFloat.h"

%include "CoreModules/Classifier.h"
%include "ClassificationModules/DTW/DTW.h"

%include "CoreModules/GestureRecognitionPipeline.h"
%include "CoreModules/PreProcessing.h"
%include "CoreModules/PostProcessing.h"
%include "CoreModules/GRTBase.h"
%include "CoreModules/FeatureExtraction.h"

%include "PreProcessingModules/MovingAverageFilter.h"

%include "FeatureExtractionModules/FFT/FastFourierTransform.h"
%include "FeatureExtractionModules/FFT/FFT.h"

%include "PostProcessingModules/ClassLabelFilter.h"
%include "PostProcessingModules/ClassLabelChangeFilter.h"
%include "PostProcessingModules/ClassLabelTimeoutFilter.h"
```

Appendix 2

Class diagrams of the three classes developed.

```
public class MainActivity
private SensorManager sensorManager;
private Sensor accelerometer;
private TextView textX;
private TextView textTrain;
private TextView textGesture;
private TextView textTitle;
private TextView textSubTitle;
private boolean recording;
private boolean gestureIt;
private boolean isTrained;
private GraphView graph;
private List<Float> valuesX;
private List<Float> valuesY;
private List<Float> valuesZ;
private LineGraphSeries<DataPoint> seriesX;
private LineGraphSeries<DataPoint> seriesY;
private LineGraphSeries<DataPoint> seriesZ;
private final GestureRecognitionPipeline;
public final String MyPREFERENCES = "MyPrefs";
private SharedPreferences preferences;
SharedPreferences.Editor editor;
private OSCPortOut oscPortOut;
private String myIP;
private int myPort;
private boolean sendOSCMessage = false;
private String addressOSC;
private  ArrayList<Object> messageOSCArray = new ArrayList<Object>();
private float likelihoodThres;
protected void onCreate(Bundle savedInstanceState)
private OSCMessage writeOSCMessage()
private void writeContentOSCMessage(int gestureLabel)
private void initPreferences()
private String getPreferenceS(String name)
private int getPreferenceI(String name)
private void newGestureJ()
private int trainGestureJ()
private void startRecordingData()
private void stopRecordingData()
private void gestureItJ()
private void drawPlot()
private void redrawPlot(List<Float> valuesX, List<Float> valuesY, List<Float> valuesZ)
private void clearPlot()
public void onSensorChanged(SensorEvent event)
public void onAccuracyChanged(Sensor sensor, int accuracy)
/* Native – Java  communication functions */
public native void readAccelerometerData(float accX, float accY, float accZ);
public native void stopRecording();
public native void skipLastRecordedGesture();
public native int trainGesture();
public native void newGesture();
public native int gestureIt(float accX, float accY, float accZ);
public native int getMaxClassLabel();
public native float getLikelihoodLastGesture();
public native float getLikelihoodThres();
```

public class Main2Activity
private long timeoutValue private float nullRejectionCoeffValue private float d = 100 private float trimThresValue private float trimPercentValue private float likelihoodValue public static final String MyPREFERENCES = "MyPrefs" private SharedPreferences preferences SharedPreferences.Editor editor private EditText editIP private EditText editPort

protected void onCreate(Bundle savedInstanceState) {
private String getPreferenceS(String name){
private int getPreferenceI(String name){
private void setPreferenceS (String name, String value){
private void setPreferenceI (String name, int value) {
/* Native – Java communication functions */
public native void setTimeout(long newCoeff)
public native long getTimeout()
public native void setNullRejectionCoeff(float newCoeff)
public native float getNullRejectionCoeff()
public native void setTrimThreshold(float newTrimThres)
public native float getTrimThreshold()
public native void setTrimPercent(float newTrimPercent)
public native float getTrimPercent()
public native void setLikelihoodThres(float newThres)
public native float getLikelihoodThres()

class Gesture
int dimensions = 3 GestureRecognitionPipeline pipeline TimeSeriesClassificationData trainingData DTW dtw UINT maxClassLabel MatrixFloat trainingSample float trimThres = 0.1 float trimPercent = 50 unsigned long timeOut = 1000 float likelihoodThreshold = 60
Gesture() int getDimensions() UINT getPredictedClassLabel() Float getMaximumLikelihood() void setTrainingSet(VectorFloat sample) void setTimeout(unsigned long newTime) unsigned long getTimeout() void setNullRejectionCoeff(float newCoeff) float getNullRejectionCoeff() void setTrimThreshold(float newThres) float getTrimThreshold() void setTrimPercent(float newPercent) float getTrimPercent() float getLikelihoodThres() void setLikelihoodThres(float newThres) UINT getMaxClassLabel() void enableTrimTrainingSamples() void saveAndResetTrainingSample() bool trainGesture() void updateMaxClassLabel() void removeLastSample() bool gotTrained() bool predict(VectorFloat sample)

Other functions developed on the Native Library

Communication between Native Library and MainActivity

```
void Java_com_luisjoglar_gestureit_MainActivity_readAccelerometerData(JNIEnv *env,
float dummy, float accX, float accY, float accZ)
void Java_com_luisjoglar_gestureit_MainActivity_stopRecording(JNIEnv *env)
void Java_com_luisjoglar_gestureit_MainActivity_skipLastRecordedGesture(JNIEnv
*env)
int Java_com_luisjoglar_gestureit_MainActivity_trainGesture(JNIEnv *env)
void Java_com_luisjoglar_gestureit_MainActivity_newGesture(JNIEnv *env)
int Java_com_luisjoglar_gestureit_MainActivity_gestureIt(JNIEnv *env, float dummy,
float accX, float accY, float accZ)
int Java_com_luisjoglar_gestureit_MainActivity_getMaxClassLabel()
float Java_com_luisjoglar_gestureit_MainActivity_getLikelihoodLastGesture(JNIEnv
*env)
float Java_com_luisjoglar_gestureit_MainActivity_getLikelihoodThres(JNIEnv *env)
```

Communication between Native Library and Main2Activity

```
void Java_com_luisjoglar_gestureit_Main2Activity_setTimeout(JNIEnv *env, float
dummy,  unsigned long newCoeff)
unsigned long Java_com_luisjoglar_gestureit_Main2Activity_getTimeout(JNIEnv *env)
void Java_com_luisjoglar_gestureit_Main2Activity_setNullRejectionCoeff(JNIEnv *env,
float dummy,  float newCoeff)
float Java_com_luisjoglar_gestureit_Main2Activity_getNullRejectionCoeff(JNIEnv
*env)
void Java_com_luisjoglar_gestureit_Main2Activity_setTrimThreshold(JNIEnv *env,
float dummy,  float newThres)
float Java_com_luisjoglar_gestureit_Main2Activity_getTrimThreshold(JNIEnv *env)
void Java_com_luisjoglar_gestureit_Main2Activity_setTrimPercent(JNIEnv *env, float
dummy,  float newPercent)
float Java_com_luisjoglar_gestureit_Main2Activity_getTrimPercent(JNIEnv *env)
void Java_com_luisjoglar_gestureit_Main2Activity_setLikelihoodThres(JNIEnv *env,
float dummy,  float newThres)
float Java_com_luisjoglar_gestureit_Main2Activity_getLikelihoodThres(JNIEnv *env)
```