Universidade de Aveiro
2022

BERNARDO INÁCIO CASTANHEIRA PEREIRA SOARES DA COSTA

Aplicação para smartphone 'Practice As You Walk': mobile learning e gamification em ensaio coral

'Practice As You Walk': a smartphone application for mobile learning and gamification in choral practice

Universidade de Aveiro
2022

BERNARDO INÁCIO CASTANHEIRA PEREIRA SOARES DA COSTA

Aplicação para smartphone 'Practice As You Walk': mobile learning e gamification em ensaio coral

'Practice As You Walk': a smartphone application for mobile learning and gamification in choral practice

Dissertação apresentada à Universidade de Aveiro para cumprimento dos requisitos necessários à obtenção do grau de Mestre em Engenharia Eletrónica e Telecomunicações, realizada sob a orientação científica do Doutor António Guilherme Campos, Professor auxiliar do Departamento de Eletrónica, Telecomunicações e Informática da Universidade de Aveiro.

o júri / the jury

presidente / president

Prof. Doutor Telmo Reis Cunha
professor associado do Departamento de Eletrónica, Telecomunicações e Informática
da Universidade de Aveiro

vogais / examiners committee

Prof. Doutor Rui Luís Nogueira Penha
professor adjunto da Escola Superior de Música e Artes do Espetáculo
do Instituto Politécnico do Porto (arguente principal)

Prof. Doutor António Guilherme Rocha Campos
professor auxiliar do Departamento de Eletrónica, Telecomunicações e Informática
da Universidade de Aveiro (orientador)

agradecimentos / acknowledgements

Aos meus pais, Sandra e Alberto, agradeço toda a paciência e apoio, com os quais sempre pude contar, e agradeço ainda por esta e por todas as outras conquistas de que me possa orgulhar.

Aos meus restantes familiares que, felizmente, são muitos, sendo difíceis de enumerar em poucas linhas, e também aos que já cá não estão, porque em mim estarão sempre, um muito obrigado.

À Emília (Mimi), agradeço pelo constante zelo pelo meu sucesso e por me tratar, desde que me recordo, como um filho.

À minha namorada, Cláudia, agradeço por todo o carinho, pelo apoio nos momentos decisivos e por me ensinar todos os dias a ser uma pessoa melhor, bem como aos seus pais, Sr. Carlos e D. Teresa, por fazerem parte da minha vida.

Ao Francisco Oliveira, por todos os momentos em que nos vimos irmãos e não só amigos, na música e fora dela, um muito obrigado.

À Academia de Música de Santa Maria da Feira e à Banda Marcial do Vale, as duas instituições a quem devo grande parte das minhas capacidades musicais, um muito obrigado.

Agradeço aos meus colegas Ivo Rodrigues e Fábio Almeida pela colaboração no desenvolvimento de algoritmos para a deteção de passo, no contexto da unidade curricular de Fundamentos de Aprendizagem Automática, lecionada pela professora Petia Georgieva, a quem agradeço por todos os esclarecimentos e sugestões.

Ao Éric Lana, pela amabilidade e constante apoio no desenvolvimento deste trabalho, e ao coro por este dirigido, os Canarinhos de Itabirito, um muito obrigado.

Aos professores Gilvano Dalagna e Clarissa Foletto, pela colaboração e pela oportunidade de participar na criação do projeto exploratório 'Ultreia', um muito obrigado.

Agradeço, ainda, ao meu colega não só de curso mas também de casa, durante os meus anos de Universidade, Manuel Silva, por todos os momentos que partilhámos, pela paciência e pela amizade.

Aos meus restantes amigos e colegas de curso, que de alguma forma ou de outra deixaram a sua marca no meu percurso, um muito obrigado.

Por último, gostaria de agradecer ao professor Guilherme Campos, não só pelo seu papel enquanto meu orientador mas por todas as conversas informais, por toda a ajuda e conhecimento dispensados desde o meu primeiro ano no MIEET, por todas as oportunidades e portas que graças a ele vi abrir e, acima de tudo, por acreditar nos seus próprios princípios e zelar, humilde e verdadeiramente, pelo sucesso dos seus alunos e colegas de profissão. O meu muito obrigado.

Palavras Chave

deteção de passo, MIDI, Unity, Android, sensores, música, sincronização, ritmo.

Resumo

Com a massificação global dos dispositivos móveis, o uso da tecnologia para fins pedagógicos no contexto da aprendizagem musical revelou ser uma ferramenta imprescindível para assegurar a motivação dos estudantes. Explorando a implementação dos conceitos de gamification e mobile learning no ensino da música, referindo casos de estudo relevantes neste campo, esta dissertação culmina com o desenvolvimento de uma aplicação para smartphones Android denominada 'Practice As You Walk'. Conforme indica o nome, esta ferramenta de aprendizagem consiste na reprodução de excertos musicais ao ritmo determinado pelo passo do utilizador, sendo uma forma de prática musical que estimula no indivíduo a memorização de obras musicais e a capacidade de sincronização. Com vista ao desenvolvimento da aplicação, são explorados neste trabalho métodos inovadores utilizados na deteção de passo através de sensores incorporados nos dispositivos móveis, tais como o acelerómetro e o giroscópio, e também apresentados os fundamentos do protocolo de comunicação MIDI para a transmissão digital de eventos relacionados com a interpretação musical. São propostos dois métodos para a deteção de passo com recurso a um smartphone, tendo o método baseado em regras atingido um F1-score de 99% e o método baseado em aprendizagem automática um F1-score de 95.84%. O desenvolvimento da aplicação, inicialmente na plataforma Unity, consiste na integração de classes para a manipulação e processamento de ficheiros MIDI com a capacidade de leitura e reprodução dos mesmos ao ritmo do passo do utilizador. Devido a alguns defeitos identificados no mecanismo de reprodução, segue-se a migração para o ambiente de desenvolvimento Android Studio recorrendo a uma biblioteca que integra o sintetizador Sonivox EAS. Esta abstração do mecanismo de reprodução permite a integração direta das funcionalidades desenvolvidas em Unity e um maior foco na construção de uma interface de utilizador cativante. Por fim, no âmbito pedagógico deste trabalho, a aplicação foi testada por membros de um coro infanto-juvenil. Um questionário revelou satisfação geral com a aplicação e permitiu a recolha de opiniões e sugestões tendo em vista potenciais melhorias.

Keywords step detection, MIDI, Unity, Android, sensors, music, entrainment, rhythm.

Abstract

With the worldwide massification of mobile devices, the use of technology for pedagogical purposes in the context of music learning has proven to be an indispensable tool for ensuring motivation among students. By exploring the implementation of concepts such as gamification and mobile learning in music education, mentioning relevant case studies in this field, this dissertation culminates with the development of an application for Android smartphones entitled 'Practice As You Walk'. As the name implies, this learning tool consists in the reproduction of musical excerpts at the user's walking pace, being a form of musical practice that stimulates the memorization of music pieces and synchronization ability of the individual. For the development of the application, this work explores innovative methods used for step detection through integrated sensors on mobile devices, such as the accelerometer and the gyroscope, and also presents the fundamentals of the MIDI communication protocol for the digital transmission of events related with musical performance. Two methods for smartphone-based step detection are proposed, with the rule-based method attaining an F1-score of 99% and the machine learning method attaining an F1-score of 95.84%. The development of the application, initially in the Unity platform, consists of integrating classes for MIDI file manipulation and processing with the ability to interpret and reproduce them at the user's walking pace. Due to some faults identified in the music playback mechanism, migration to the Android Studio IDE took place through a third-party library that integrates the Sonivox EAS synthesizer. This abstraction from the playback mechanism allowed direct incorporation of the core functionalities developed in Unity and focus on the construction of a captivating user interface. Finally, within the pedagogical purpose of the present work, the application was tested by members of a children and youth choir. The questionnaire revealed general satisfaction with the application, allowing collection of opinions and suggestions on potential future improvements.

Contents

Contents i

List of Figures v

List of Tables vii

Glossary ix

1. Introduction 1

1.1 Background . . . 1

1.2 Motivation . . . 2

1.3 Objectives . . . 2

1.4 Contributions . . . 3

1.5 Dissertation structure . . . 3

2. Technology in music education 5

2.1 Background . . . 5

2.2 Advances in music pedagogy . . . 5

2.2.1 Mobile learning . . . 5

2.2.2 Gamification . . . 6

2.2.3 Underlying challenges . . . 6

2.2.4 Related case studies . . . 7

2.3 'Practice As You Walk' . . . 8

2.3.1 Mission . . . 8

2.3.2 Educational vs. recreational usage . . . 8

2.3.3 Similar work . . . 8

3. Smartphone-based step detection 9

3.1 Motion sensors in smartphones . . . 9
3.1.1 Inertial measurement unit (IMU) . . . 9
3.1.2 Accelerometer and gyroscope . . . 9
3.2 Advances in smartphone-based step detection . . . 10
3.2.1 Fundamental concepts . . . 10
3.2.2 Selected literature examples . . . 11
3.3 Development background . . . 12
3.3.1 Defined constraints . . . 12
3.3.2 Environment and methodology . . . 12
3.4 The Unity platform . . . 13
3.4.1 Brief introduction . . . 13
3.4.2 Editor interface . . . 13
3.4.3 Gameplay elements . . . 13
3.4.4 Building for Android . . . 14
3.4.5 The *Input* class . . . 15
3.5 Data collection . . . 15
3.6 Rule-based approach . . . 17
3.6.1 Raw data . . . 17
3.6.2 GT correction . . . 18
3.6.3 Low-pass filtering . . . 19
3.6.4 Training and testing sets . . . 19
3.6.5 Detection algorithm . . . 19
3.6.6 Classifier performance metrics . . . 20
3.6.7 Threshold refinement and attained results . . . 20
3.7 Machine Learning approach . . . 21
3.7.1 Support Vector Machine (SVM) . . . 21
3.7.2 Implementation . . . 23
3.7.3 Performance on testing set . . . 26
3.8 Conclusions . . . 26

4. Application development **27**
4.1 *Modus operandi* . . . 27
4.1.1 Extracting tempo information from walking . . . 27
4.1.2 Music playback . . . 28
4.1.3 Achieving entrainment . . . 28
4.2 Initial prototype . . . 28
4.3 Brief overview of the MIDI standard . . . 29
4.3.1 Introduction . . . 29

4.3.2 Hardware specification . . . 29
4.3.3 MIDI messages . . . 29
4.3.4 *Running Status* . . . 29
4.3.5 Standard MIDI File (SMF) . . . 30
4.3.6 Additional information . . . 30
4.4 Development in Unity . . . 30
4.4.1 Third-party libraries . . . 30
4.4.2 User interface . . . 31
4.4.3 Internal structure and operation . . . 31
4.4.4 Identified problems . . . 33
4.4.5 Conclusions . . . 33
4.5 Development in Android Studio . . . 34
4.5.1 Overview . . . 34
4.5.2 Third-party libraries . . . 34
4.5.3 Retaining functionalities from the Unity app . . . 35
4.5.4 User interface . . . 35
4.5.5 Main implementation differences . . . 35

5. Validation and discussion **37**
5.1 Final version of the Android application . . . 37
5.2 Validation . . . 38
5.2.1 Test group . . . 38
5.2.2 Test procedure . . . 38
5.2.3 Questionnaire . . . 38
5.2.4 Results . . . 38
5.2.5 Future work . . . 39

Appendix A: C♯ code **41**

Appendix B: MATLAB code **45**

Appendix C: Java code **49**

Appendix D: Questionnaire (in Portuguese) **55**

References **63**

This page intentionally left blank.

List of Figures

1.1 Number of smartphones sold worldwide from 2007 to 2021 (in million units) 1

3.1 Gyroscope (L3G4200D) and accelerometer (LIS331DLH) sensors 9
3.2 Pitch, roll, and yaw relative to the initial reference frame 10
3.3 Typical layout of the Unity Editor 13
3.4 Switching Unity build platform to Android 14
3.5 Walking while holding the smartphone in the texting position 15
3.6 Acquired accelerometer data from the three users (MATLAB) 16
3.7 Acquired gyroscope data from the three users (MATLAB) 16
3.8 Comparison between light screen tapping and walking signals (MATLAB) 17
3.9 Cropped screen capture of the mobile app for data collection 17
3.10 Raw accelerometer data and GT (MATLAB) 18
3.11 Corrected GT, matching accelerometer data (MATLAB) 18
3.12 Filtered accelerometer data (MATLAB) 19
3.13 Proposed rule-based step detection algorithm 19
3.14 Visual representation of the SVM algorithm 21
3.15 Large margin cost functions for $y = 1$ and $y = 0$ 22
3.16 Step and non-step examples 24
3.17 CV error evolution for the four SVM kernels 25

4.1 Extracting playback tempo from the user's walking pace 27
4.2 Simple depiction of the internal music playback mechanism 28
4.3 MIDI *Note On* message 29
4.4 Categories of MIDI messages 29
4.5 User interface of the app developed in Unity 31
4.6 Audio settings for the Unity project 33
4.7 Typical layout of Android Studio 34

4.8 User interface of the app developed in Android Studio 35

5.1 Application: splash screen (left) and interface (right) 37

List of Tables

3.1 Confusion matrices for the rule-based classifier 21
3.2 Best performing kernel and window sizes 24
3.3 Refinement of SVM degree and parameters 25
3.4 Confusion matrix for the SVM classifier 26

This page intentionally left blank.

Glossary

AI	Artificial Intelligence
API	Application Programming Interface
BPM	Beats Per Minute
CC	Control Change
CV	Cross-Validation
DL	Deep Learning
GUI	Graphical User Interface
GT	Ground Truth
IC	Integrated Circuit
IDE	Integrated Development Environment
IMU	Inertial Measurement Unit
MEMS	Microelectromechanical Sensor
MIDI	Musical Instrument Digital Interface
ML	Machine Learning
mLearning	Mobile Learning
PPQN	Pulses Per Quarter Note
RBF	Radial Basis Function
SMF	Standard MIDI File
SPM	Steps Per Minute
SVM	Support Vector Machine

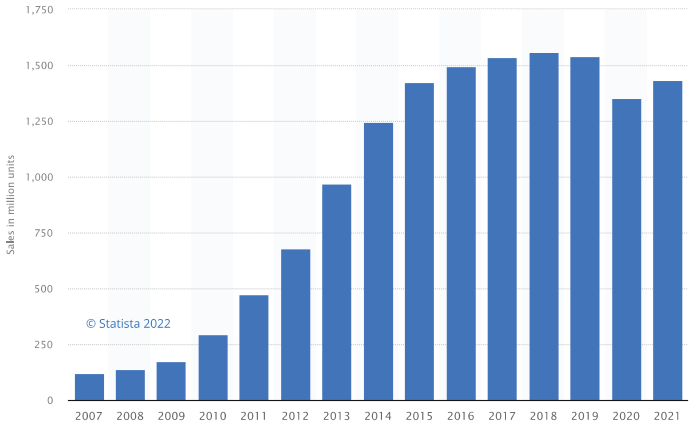
This page intentionally left blank.

Chapter 1

Introduction

1.1 Background

The massification of mobile devices, especially since the beginning of the 21st century (see Figure 1.1), is currently of the most controversial topics regarding its effects on society: while effectively the dominant means of communication and information dissemination, smartphones have been considered the source of many problems among younger generations, such as loneliness [1], family estrangement [2], procrastination [3] and distraction in the classroom [4].



Source: `https://www.statista.com/statistics/263437/global-smartphone-sales-to-end-users-since-2007`

Figure 1.1: Number of smartphones sold worldwide from 2007 to 2021 (in million units)

While there is no immediate solution to every stated issue, people are welcome to give their contribution in order to attenuate them; for instance, mobile learning (mLearning) and gamification are the result of decades of work dedicated to improving the learning experience through the use of technology, turning smartphones into an important part of the classroom.

Therefore, in spite of the currently faced disadvantages regarding the usage of mobile devices, it is our duty, if and when possible, to try and incorporate modern technology as means of expanding our knowledge in a fun, interactive and engaging manner. This strategy is becoming more and more of a standard in many classrooms [5], whether through plain online availability of class resources or through more complex activities, such as games and quizzes.

Naturally, music educators around the world also began adopting mobile devices as powerful interactive learning tools. Researchers have reported the positive impacts of this approach, from teachers' technology acceptance to students' increased motivation and performance, but also indicate that mobile devices still remain underused in music learning [6]. This is especially true regarding choral practice: while there are many studies linking music and technology, only a few establish a strong connection between choral singing and mobile applications [7].

1.2 Motivation

The present work arises from an ongoing doctoral project in Music at the Department of Communication and Art (DeCA) of the University of Aveiro, entitled 'Práticas de Direcção em Coros Infanto-Juvenis: Abordagens Pedagógicas e Recursos M-Learning' (Lana, É.).

Starting from a pilot study that underlines the potential advantages of mLearning as a support tool for children and youth choral performance, this research project aims to develop new forms of choral practice and undertake mLearning, considering innovative learning ecologies, as constitutive elements of pedagogical approaches applied to children and youth.

The fact that the bonds established by teenagers with music are increasingly associated with the use of mobile computing devices [8] can be explored in the context of children and youth choral practice, where mLearning resources may succeed in promoting or stimulating:

- autonomous music learning;
- concentration over long periods of time;
- informal and/or collaborative learning experiences;
- communication and interaction between choir and director;
- identification of strengths/weaknesses;
- self-esteem and self-confidence boosts.

Accordingly, sculpting the paradigm of interactive games for didactic purposes, which is a short but unassailable description of gamification, can provide many educational benefits [9].

1.3 Objectives

The main purpose of this work is to develop an Android smartphone application, entitled 'Practice As You Walk', for autonomous and informal learning through the following features:

- importing external files corresponding to musical excerpts;
- choosing certain melodic/rhythmic lines in a given excerpt;
- continuously extracting tempo from the user's walking pace;
- continuously updating music playback with the extracted tempo.

Studies show that music as a background phenomenon can influence the way humans walk in an unconscious way, even in individuals with no musical proficiency [10]. The idea is to explore this intrinsic relation between walking and music playback with the user as an active listener, who unconsciously starts memorizing melodic lines and attaining rhythmic dexterity.

Evidently, the development phase is preceded by an extensive period of research in all constitutive areas, which results in adjustments and constraints to the initial specifications.

1.4 Contributions

The ongoing doctoral project mentioned in section 1.2, whose author (Lana, É.) is an active participant in both development and validation of 'Practice As You Walk', has significant focus on mLearning as a pedagogical approach. Therefore, the end results of this collaboration are critical to both parts, especially for understanding how well the target audience interacts with the application and how much of this interaction actually serves a didactic purpose.

An initial prototype of 'Practice As You Walk', rapidly built during the first weeks of research, led to the interest of some researchers at DeCA to undertake an exploratory project on the role of the audience in artistic creation, heavily involving the research done on the main topics of this work (step detection and music playback) together with virtual reality.

Entitled 'Create As You Walk' (Dalagna, G., Foletto, C., Campos, G. and Costa, B.), this project envisions the construction of an installation artwork where the user, as creator of the audiovisual landscape, can experience the Camino de Santiago routes in virtual reality [11].

1.5 Dissertation structure

The present document is organized in five chapters, ordered according to the natural progress of this work, from the initial research to the actual development of the application.

Right after the introductory chapter, the reader is made familiar with the field of music education (Chapter 2), where fundamental concepts and typical approaches to music pedagogy are discussed, along with the use of technology for the learning process; the present work is observed from a pedagogical perspective and compared with existing work of the same nature.

The following two chapters introduce the reader to the main challenge of this work: integrating real-time step detection with music playback. Chapter 3 comprises a literature review on step detection, presenting multiple techniques, each with their own advantages and disadvantages, and discussing their applicability in the present work; then, a comprehensive approach to the typical sensors found in smartphones takes place and applicable step detection techniques are discussed and implemented. The Unity platform is introduced here.

Then, in Chapter 4, the full application development process is described: starting with the principle of operation, which is the articulation between the step detection algorithm and the music playback mechanism, this chapter reports the creation of an initial prototype and introduces the Musical Instrument Digital Interface (MIDI) standard and related music/audio terminology, followed by the first version of the application developed in Unity, with mentions to each third-party library used, and associated problems. Then, the subsequent adoption of the Android Studio integrated development environment (IDE) is discussed, as well as some final remarks on the assembled solution. The validation of the finished application is presented in Chapter 5, which analyses a questionnaire presented to a select test group; then, possible improvements and additional functionalities to the application are discussed.

This page intentionally left blank.

Chapter 2

Technology in music education

2.1 Background

Studies show that music education is undergoing an internal crisis triggered by the decay of the educational institution. Among others, this is due to the following reasons [12] [13]:

- music education is part of a curricular system regulated by technocratic interests;
- generally, public schools lack adequate infrastructure for music education;
- commercial music has a growing impact on school culture;
- parents and peers discourage young people from attending music classes;
- music education is associated with a low curriculum status.

Therefore, educational institutions should be capable of adapting to the social contexts to which they belong [12]: a transition to the digital classroom through mobile devices is crucial.

2.2 Advances in music pedagogy

Recent advances in technologies for human-computer interaction, especially mobile devices such as smartphones, tablets or handheld consoles, are useful tools to keep students interested and motivated, for improving their cognitive abilities and reinforcing the learning process [14].

2.2.1 Mobile learning

A growing concept among younger generations, mLearning is the ability to achieve or provide educational content, in the form of digital learning resources, on typical mobile devices; users can access, share or create content both inside and outside of the classroom [15].

Some of the benefits of mLearning [16] are:

- improved retention, performance, motivation and autonomy;
- attractive, self-directed and informal learning;
- portability and accessibility of the educational technology;
- innovative pedagogical approaches for teachers;
- enhanced teacher-student and student-student communication;
- ubiquitous learning process with sense of achievement and enjoyment.

2.2.2 Gamification

In an educational context, gamification is a powerful tool for driving the attention and commitment of students by transferring game mechanics to the learning process; when enforced for educational purposes, gamification opens new doors for self-directed experimental learning, where students are hooked by fun and implicitly rewarded with knowledge and skills [17].

Advantages of gamification [18] [19] [20] include:

- recognizing the value of extended practice;
- development of personal qualities, such as creativity and persistence;
- understanding failure as an essential part of learning;
- immediate rewards for completed tasks, instead of vague long-term benefits;
- instant feedback for both teachers and students;
- improved self-efficacy and knowledge retention;
- willingness to improve instead of anxiety over bad performance;
- celebration of whole-class accomplishments, rather than individual achievements.

2.2.3 Underlying challenges

Unsurprisingly, the incorporation of mLearning and gamification into the educational system is accompanied by significant challenges, which should be identified and addressed as soon as possible, so that both of these tools can succeed within their pedagogical purposes instead of representing additional sources of detachment or only serving as entertainment.

For instance, the adoption of an mLearning scheme involves a wide range of limitations: regarding technology, aside from equipment cost, the designed platform should account for applicability in an educational context, compatibility among different devices, usability and visual quality of small devices, security and privacy issues, battery duration and connectivity restrictions; socially and culturally speaking, there is a need to overcome rejection of change and aversion towards mobile devices among educators, helping them make the transition to the digital world, while also ensuring every student has access to the available digital resources and appreciates them not as a distraction, but as another opportunity to thrive [16] [21].

Similarly, the existing literature on the effects of gamification seems to exhibit an overall bias towards its benefits, which possibly occurs either due to the lack of research on its negative impacts or the unwillingness to publish negative results. Thus, an unsettling issue arises: ignoring side-effects such as frustration, negative social comparison and decreased motivation [20] contributes to immoderate and imprudent implementations of gamification.

From a pedagogical perspective, even playing a game can feel the same as a typical class just by being mandatory. Therefore, a correct implementation must ensure that effort, instead of proficiency, is rewarded; repeatability, in the case of unsuccessful attempts, transforms failure into new opportunities; game design, while objective, does not translate to a monotonous experience, by using in-game feedback to encourage further activities, and does not rely solely on motivators, such as points and badges, that are negligible for uncompetitive students [19].

2.2.4 Related case studies

Gamification in teaching music: case study

Researchers at the University of Algarve and Universidade Aberta (Gomes, C., Figueiredo, M. and Bidarra, J.) present a paramount study [22] on how the introduction of an application affected the quality of the learning process of music education (second cycle of basic education).

In this investigation, the students could access multimedia materials, designed to provide support for recorder and guitar practice, in two ways: as a supervised activity in the classroom context (group A) or through the Moodle platform, through a game that consisted of multiple questions, each unlocking the desired materials when correctly answered (group B). A third (control) group was taught the same programmatic content, but with traditional didactic objects, a disciplinary manual in paper form and no access to the multimedia materials.

Results from an eight-week observation period show not only increased motivation in the two groups that have access to the multimedia materials, but also that group B greatly outperformed the other two in the remaining three categories: lesson preparation, overlapping areas of interest (sharing acquired competences out of class) and quality of results obtained.

Exploring a leaderboard alternative in a gamified mobile app for music learning

The use of 'PracticeCactus', a mobile app to support independent piano practice in the context of a music teacher's private studio, is explored in a conference paper [23] by researchers at Tyndale University College and the University of Toronto (Birch, H. and Brett, C.).

By gamifying independent practice through a feature known as 'Daily Practice Goal' (DPG) notifications, this application intends to create an alternative to the traditional leaderboard while avoiding isolation, common in the standard individual practice experience; a participatory design methodology took place, in which piano students and their parents collaborated in the development of the application, aside from teachers, researchers and programmers. As a result, 'PracticeCactus' revolves around three main objectives to address the loneliness felt by piano students during independent practice: sharing, quantifying and listening to practice.

When the application is active on a mobile device, it automatically detects the sound of the student practicing the piano, through an embedded acoustic analysis module. With this information, the total practice time during each 24-hour period is recorded; if the user reaches their self-determined daily goal, a DPG notification is generated on the 'Community' page of the application, where other students can see or interact with it, by liking or commenting. No information regarding actual study time is revealed to other users, since DPG notifications should not invoke comparisons, but rather highlight students' efforts inside a community. The students were free to use the application as frequently as they wanted, and were given no instructions except from how to create a 'PracticeCactus' account. Throughout a period of twenty weeks, the students, aged ten to fifteen, recorded a total of 694 practice sessions.

Seeing others reach their DPG, the students felt reassured, competitive, motivated and supportive of their peers, and explained their desire to let the community know they were practising, agreeing about the positive encouragement of 'likes' and comments on the platform.

2.3 'Practice As You Walk'

2.3.1 Mission

Within the context of mLearning and gamification in music education, 'Practice As You Walk' aims to extend practice time through the idea of synchronizing musical excerpts with the user's walking pace, constituting a portable activity that only requires a smartphone.

Implicitly, the application innovates the concept of practice, not only helping students in the rehearsal of certain musical sections, but also nurturing their body-mind relationship, as music and rhythmic movement can significantly shape the quantity and quality of behavioral responses [24]. Our internal isochrony, the ability to rhythmically divide time into equal parts and an inherent characteristic of our autonomic nervous system that is the basis of human language and musical creation [25], is responsive to external sources of rhythm [26], and is therefore subject to stimulation while 'Practice As You Walk' is in use. This is supported by the way users attempt to synchronize their pace to the music, a process known as entrainment.

2.3.2 Educational vs. recreational usage

As previously explored in subsection 2.2.3, non-pedagogical application design approaches, inappropriateness of mLearning and/or gamification for a given learning context and inadequate rewarding systems, among others, may defeat the educational benefits of modern technology. Nevertheless, 'Practice As You Walk' aims to captivate students only by positioning them as part of the music playback mechanism, making no particular distinction between educational and recreational usage, thus always serving the pedagogical purposes of active listening and memorization, which improve slowly and abstractedly through repetition.

2.3.3 Similar work

D-Jogger: Syncing music with walking

D-Jogger, from researchers at Ghent University (Moens, B., Noorden, L. and Leman, M.), is an interface that dynamically selects music and adapts its tempo to the user's pace [27].

In the reported experiment, the user is positioned on a treadmill with a wireless accelerometer attached to their left ankle, in front of a screen displaying the graphical user interface (GUI) that is connected to the computer from where the control interface operates.

If the user's pace, in steps per minute (SPM), is between 90% and 110% of the song's tempo in beats per minute (BPM), audio playback speed is changed by time stretching, with no effect on pitch; if the pace falls outside that range for more than five seconds, another song with a closer tempo is selected from the playlist. The step detection algorithm is of rule-based nature, consisting of a second-order signal interpolation at a sample rate of 200 Hz, low-pass filtering and autocorrelation of the last four seconds (aiming for the 30-125 SPM range); then, the refined value of SPM is doubled (sensor in just one leg) and averaged with previous values.

A user survey reported that achieving entrainment was stimulating and motivating.

Chapter 3

Smartphone-based step detection

3.1 Motion sensors in smartphones

Modern smartphones are typically equipped with a wide range of sensors for different purposes, such as ambient light intensity detection, fingerprint scanning or motion detection.

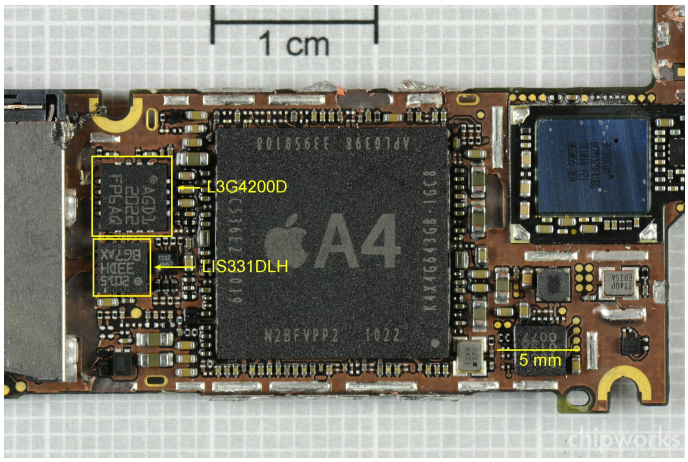
3.1.1 Inertial measurement unit (IMU)

Herein, focus is given to the built-in motion detection sensors, technically called inertial sensors, since these rely on measuring motion with respect to an inertial reference frame.

In smartphones, inertial sensors are commonly part of a microelectromechanical sensor (MEMS) IMU [28], which usually consists of three orthogonally arranged accelerometers for measuring proper acceleration, three gyroscopes for measuring angular velocity and possibly three magnetometers to sense the magnetic field surrounding the system (typically Earth's).

3.1.2 Accelerometer and gyroscope

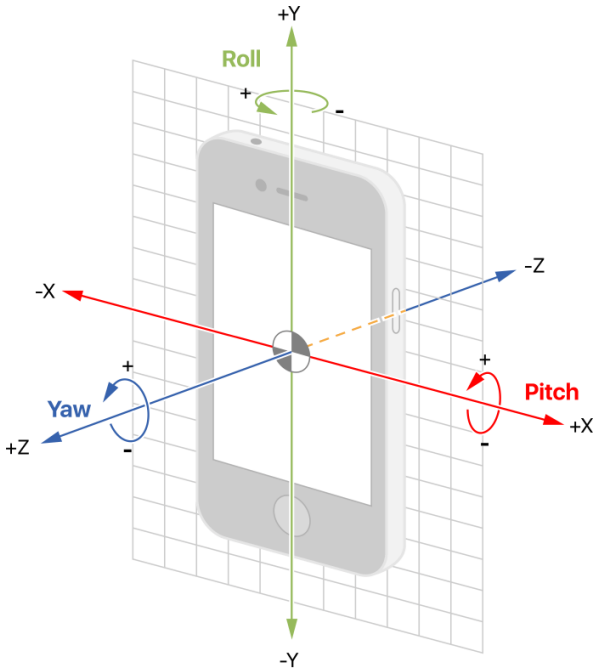
For the development of the present work, the accelerometer and gyroscope sensors are the relevant part of the IMU. Figure 3.1 identifies these sensors inside an Apple iPhone 4.



Source: `https://www.memsjournal.com/2010/12/motion-sensing-in-the-iphone-4-mems-accelerometer.html`

Figure 3.1: Gyroscope (L3G4200D) and accelerometer (LIS331DLH) sensors

As stated in subsection 3.1.1, the integrated circuits (ICs) highlighted in yellow in Figure 3.1 correspond to a triaxial accelerometer and a triaxial gyroscope, respectively. Considering the reference frame of the smartphone, these sensors measure acceleration along (and rotation around) the X axis (pitch), the Y axis (roll) and the Z axis (yaw), as shown in Figure 3.2.



Source: `https://developer.apple.com/documentation/coremotion/getting_processed_device-motion_data`

Figure 3.2: Pitch, roll, and yaw relative to the initial reference frame

3.2 Advances in smartphone-based step detection

Step detection has been a frequent research topic in the last decade, especially due to the increased interest in human activity recognition, which is useful for various technological fields such as indoor positioning systems, virtual/augmented reality and health/fitness applications.

Due to the nature of the present work, this section focuses on presenting the latest documented technological developments on step detection where a smartphone is at least part of the core system; this means that, whether built-in and/or external sensors are used, a smartphone must be involved in the foremost stages of data acquisition and/or processing.

3.2.1 Fundamental concepts

Artificial intelligence (AI):

A branch of computer science, artificial intelligence designates the development of computerized solutions to problems which otherwise would require human intelligence [29].

Modern AI-based technologies, such as spam filters, smart digital assistants, face recognition algorithms and self-driving cars, may incorporate sets of human-defined rules (rule-based methods), machine learning (ML)/deep learning (DL) algorithms or a combination of both.

Rule-based methods:

When a given system consists of a predefined relationship between its outputs and the input data (is explicitly programmed), the machine only acts according to the specified rules.

These methods are the go-to solution if there are obvious and constant regularities in data [30], which allow specifying a sequential set of operations that can produce reliable outputs.

Machine learning (ML):

As the name implies, machine learning denotes a machine's ability to learn directly from data without being explicitly programmed [31]. In contrast to rule-based methods, ML-based systems are given a specific set of data from which they learn a specific function (narrow AI).

There are two major distinct ways for ML-based systems to learn from data: supervised learning, where a label is given for each training example, and unsupervised learning, where the system must find similarities between training examples and group them accordingly.

Deep learning (DL):

For decades, ML-based systems required expertise in the domain of feature engineering, which is the process of analysing available data and transforming it into a suitable representation for the learning algorithm; advances in the field resulted in the development of methods that automatically construct those representations from raw data (representation learning).

As a subset of ML, deep learning is a term that designates complex learning methods that are composed by multiple levels of representation learning, starting from the raw input data, where the automatically learned features typically have no meaning for humans, and evolving into higher, more abstract layers of representation, until the refined features are able to emphasize distinct characteristics within the input data, ensuring correct predictions [32].

3.2.2 Selected literature examples

Classic machine learning techniques

In [33], the authors propose step detection using a smartphone accelerometer and gyroscope, considering the human gait cycle; the project, named smartSTEP, is composed of three main components: a first ML model to predict peak-step instants, a second ML model to predict valley-step instants and a smart fusion process to decide which model should be used as the main step detector, depending on the hand motion mode. Using a light gradient boosting machine (LightGBM), an open source algorithm by Microsoft based on histogram gradient boosting decision trees (GBDT), this project achieved 99% recall and 98.9% precision.

Combining the smartphone with a wearable IMU, the authors in [34] propose a K-nearest neighbors (KNN) approach for step prediction. The main focus is on the pre-processing of the acquired data: from the raw acceleration signals, a low-pass filter is applied to every axis, where strides are detected and segmented; a total of five features are extracted from both

smartphone and IMU (peak width, peak height, mean of acceleration, standard deviation of acceleration and power of acceleration). With the KNN model, a score of 91% was attained.

Other approach is presented in [35] where the authors implement step detection supporting different smartphone positions, such as handheld, texting and trouser pockets. A support vector machine (SVM) is used to identify the position of the smartphone and steps are detected via zero-crossing, through the magnitude of the accelerometer (all axes). The proposed solution returned an overall accuracy of approximately 97.1%, making this approach a robust solution independently of the speed of the pedestrian or the placement of the smartphone.

Deep learning techniques

In [36], the author proposes a convolutional neural network (CNN) for step detection using an unconstrained smartphone, through the integrated accelerometer and gyroscope sensors. The model was trained with four different smartphone positions, five step activities and two false-step activities; the testing set was afterwards provided by a random user. The ground truth was recorded via two Fraunhofer AICOS Pandlets inertial sensors, placed in the user's ankles. The smartphone data was fed to the CNN in time intervals of 256 samples from all 3 channels, the X, Y and Z axes. Then, the model was trained using batches of 32 samples, producing an output with a shape of 128 samples. An Adam optimizer with default settings was used for attaining the best results without very slow convergence. The loss function was calculated through binary cross-entropy. An overall F1-score of 88.87% was achieved.

3.3 Development background

3.3.1 Defined constraints

The adopted step detection algorithm will only collect data from integrated smartphone sensors, resulting in a cheaper, widely available and less constrained learning experience.

Since the application is later intended to reproduce musical excerpts, another constraint regarding the minimum Android version emerges: the MIDI application programming interface (API) was only released within Android 6 (API level 23), which must be taken into account.

3.3.2 Environment and methodology

The next sections of this chapter describe the development of step detection solutions, which took place during the 'Fundamentals of Machine Learning' course, taught at the Department of Electronics, Telecommunications and Informatics of the University of Aveiro.

For this purpose, the sequence of procedures below was followed:

1. creation of a Unity smartphone application for collecting data;
2. collected data pre-processing in MATLAB;
3. development of parameter-based solutions in MATLAB;
4. development of ML solutions in Jupyter Notebook (Python) using the *sci-kit* library.

3.4 The Unity platform

3.4.1 Brief introduction

Unity is a cross-platform game engine developed by Unity Technologies, which provides access to all the tools needed for development in one place: the Unity Editor. It allows users to create 2D and 3D games, apps and experiences, and offers a scripting API in C♯ using Mono, an open source development platform based on the Microsoft .NET Framework [37].

3.4.2 Editor interface

A common arrangement of the windows inside Unity Editor is shown in Figure 3.3.

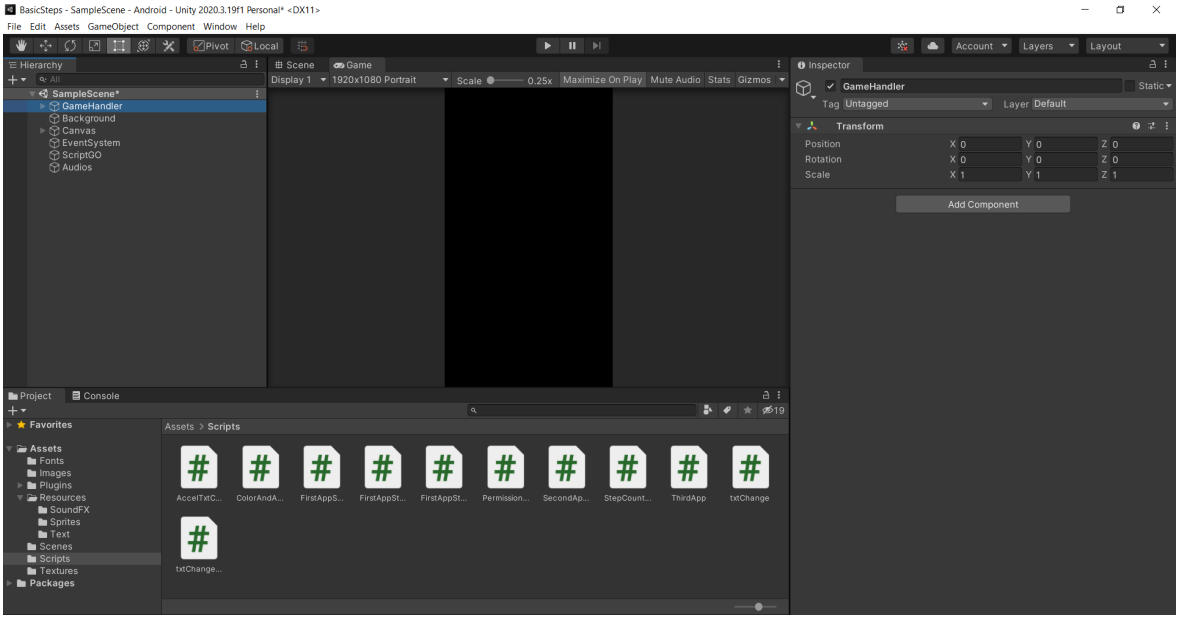


Figure 3.3: Typical layout of the Unity Editor

3.4.3 Gameplay elements

The various elements of the Unity Editor are optimally designed for game development; the following subsections provide a condensed explanation [37] of each element employed.

Scenes:

Scenes contain all or part of a certain application; for example, a simple game can be built using a single *Scene*, while having one *Scene* per level might be useful in more complex games.

Each *Scene* can have its own environments and objects, as listed in the *Hierarchy* window.

GameObjects:

Every object in a Unity app is a *GameObject*, the core structure of the editor; whether a *Camera*, a *Light* or a menu, their functionality is defined by *Components* attached to them.

Components:

Components are the functional pieces of *GameObjects*, containing editable properties that define their behavior; some examples are *Cameras*, *Audio Sources* and *Scripts*, among others.

Cameras:

GameObjects in a Unity *Scene* are represented in a three-dimensional space; as device screens are two-dimensional, *Cameras* can capture a given view and flatten it for display.

Audio Sources:

In a certain *Scene*, an *Audio Source* is responsible for playing back an *Audio Clip*. Multiple configurations are available, ranging from *Volume*, *Pitch* and *Reverb* to *3D Sound Settings*.

Scripts:

C♯ *Scripts* are *Assets* used to trigger game events or modify *GameObjects* during execution.

Assets:

Any items used within an Unity project, ranging from text files to *Audio Clips*, are *Assets*.

3.4.4 Building for Android

For Android smartphone development, the resolution should be set to a typical smartphone (e.g. 9:16) ratio, with or without the possibility for orientation changes (e.g. to 16:9, widescreen version), and the destination (build) platform should be set to Android, as shown below:

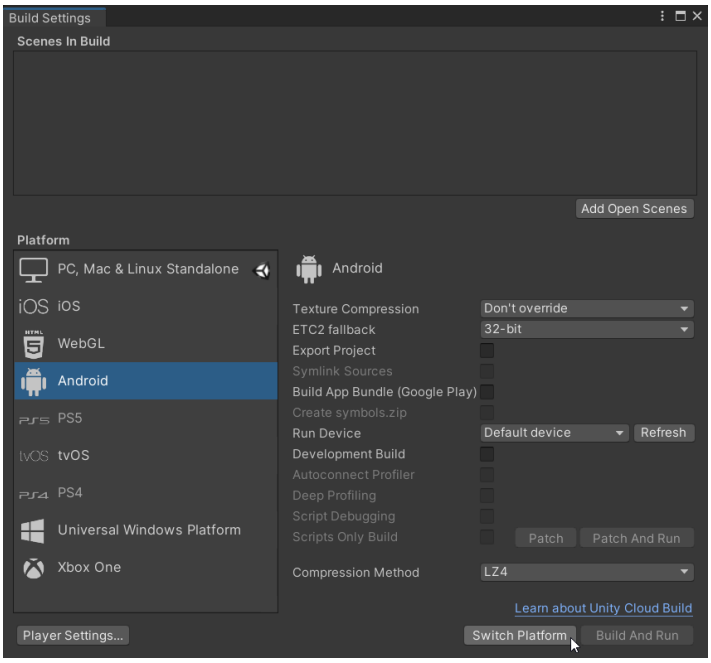


Figure 3.4: Switching Unity build platform to Android

3.4.5 The *Input* class

As the main *Input System* interface, this class allows control of the application through devices, built-in sensors, touch or gestures, and supports multiple devices [37], such as:

- mice and keyboards;
- joysticks;
- touch screens;
- sensor data on mobile devices;
- virtual/augmented reality (VR/AR) controllers.

When a device moves, its IMU reports acceleration and rotation changes along the three primary axes in three-dimensional space; for instance, acceleration along each axis (as shown in Figure 3.2) is reported as a g-force value, obtained using the *Input.acceleration* property.

3.5 Data collection

A small Unity smartphone application was developed for collecting sensor data (accelerometer and gyroscope) and ground truth (GT) (step/non-step labels), saving it into text files.

Having in mind the possibility of future app users to train their own step detection models in a simple way, the user should click a button in the interface as close as possible to the time their feet hit the floor, so as to report a step event. However, this is only feasible if touching the screen does not significantly impact the sensor signals; for this, two constraints arose:

- the smartphone should be held in the texting position, as shown in Figure 3.5;
- screen touch sensitivity should be maximized, so that very light taps are considered.

Source: `https://www.needpix.com/photo/1149112`

Figure 3.5: Walking while holding the smartphone in the texting position

Using a Samsung Galaxy A80 smartphone, two preliminary tests were carried in order to better understand the acquired sensor data and the validity of the chosen methodology:

- collecting sensor data while walking;
- collecting sensor data while lightly tapping the screen.

A comparison between accelerometer (Figure 3.6) and gyroscope (Figure 3.7) signals, for three individuals of different heights and weights walking on the same surface, revealed magnitude similarity between the three signals and that screen taps are negligible (Figure 3.8).

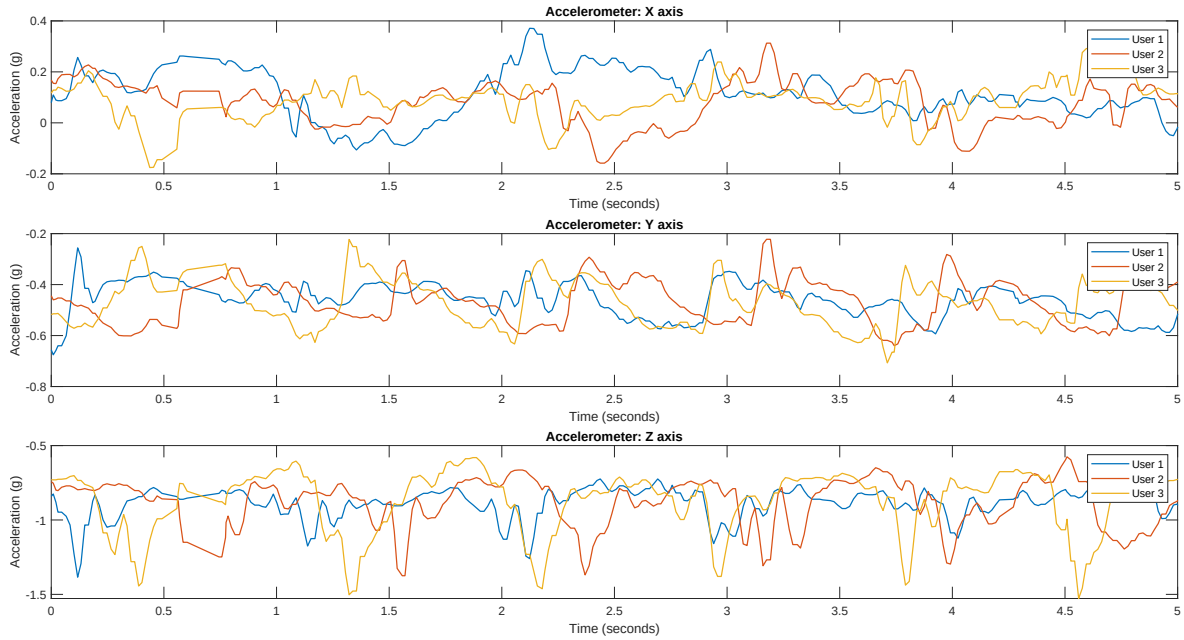


Figure 3.6: Acquired accelerometer data from the three users (MATLAB)

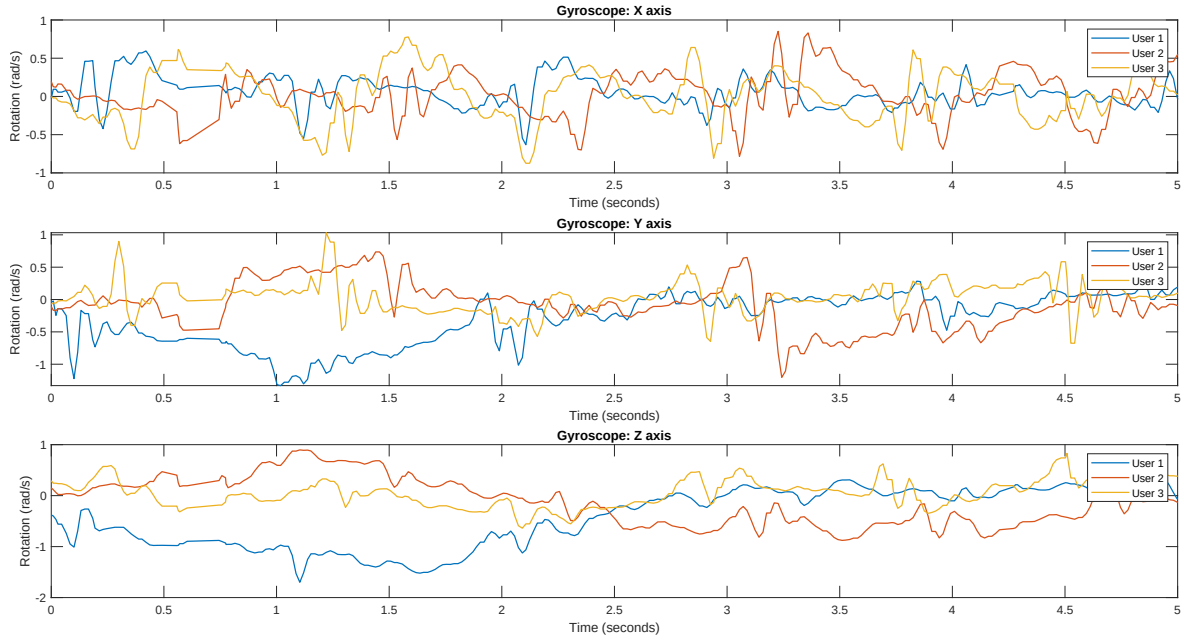


Figure 3.7: Acquired gyroscope data from the three users (MATLAB)

Code for the implementation of this application is in Appendix A, section A.1; the sampling rate, according to the script, is the same as the device's frame rate (60 Hz for a Galaxy A80).

Additional information can be extracted from these graphs, considering the axes in Figure 3.2: as expected, using the texting position, smartphone acceleration along the X axis does not provide useful information for step detection; similarly, the only significant rotation is around the X axis, but it still shows many undesired fluctuations and user dependency.

Since light screen taps do not significantly affect sensor signals, the ability to label step occurrences is added to the application (also included in Appendix A, section A.2).

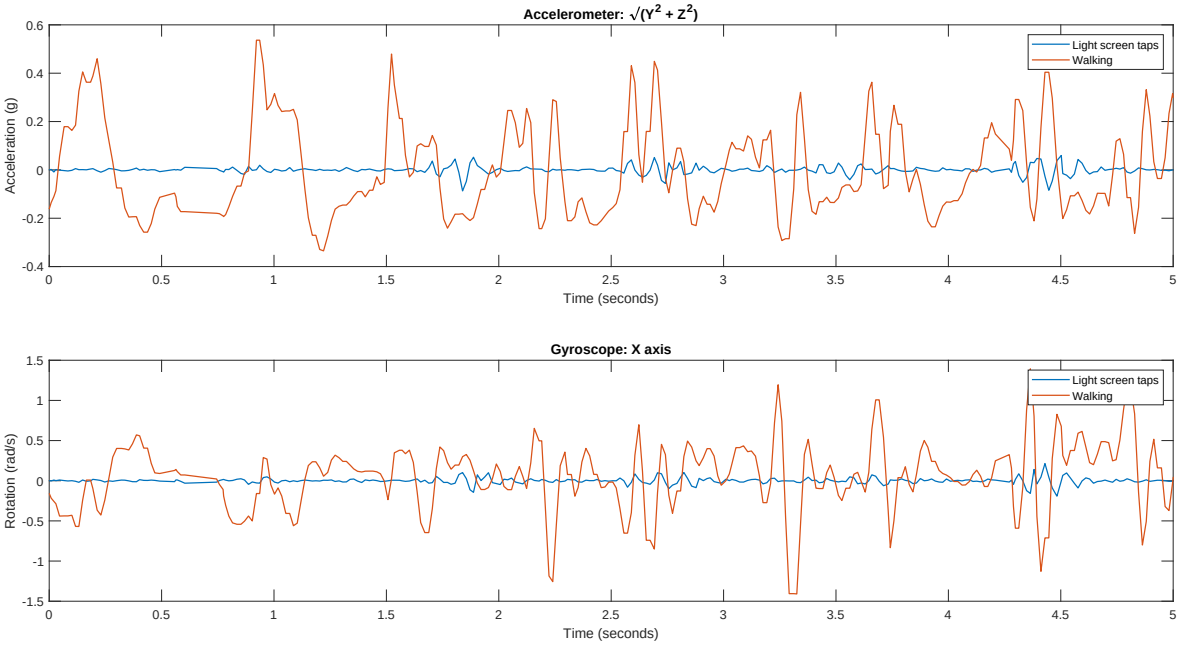


Figure 3.8: Comparison between light screen tapping and walking signals (MATLAB)

With the added features, as shown in Figure 3.9, the GT is now provided by the user.

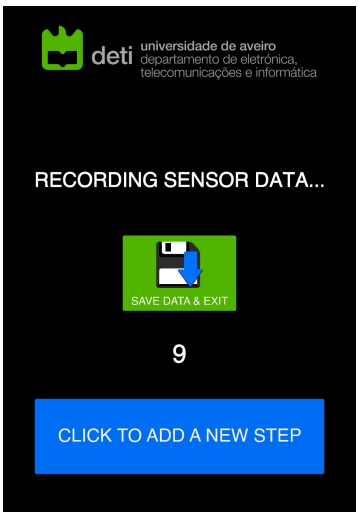


Figure 3.9: Cropped screen capture of the mobile app for data collection

3.6 Rule-based approach

3.6.1 Raw data

Considering the remarks made in the last section, the data to be collected is the acceleration along the Y and Z axes of the device; rotation around the X axis is not considered, for now.

The user presses the button approximately when their feet hit the floor, signaling a step event, which returns a single '1' to the GT vector, with non-steps represented by '0'. The raw accelerometer data and the GT vector provided by the user can be observed in Figure 3.10.

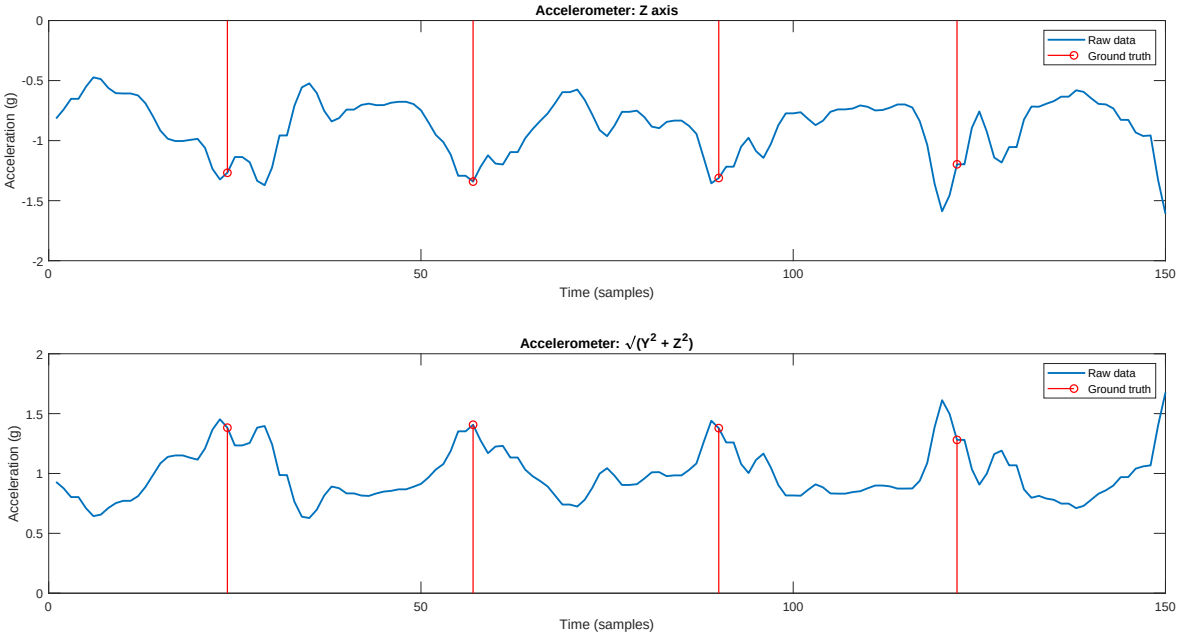


Figure 3.10: Raw accelerometer data and GT (MATLAB)

Since the texting position maintains an angle of around 45° between the device and the floor, the contribution of both Y and Z axes to the vertical force is represented by $\sqrt{Y^2 + Z^2}$.

3.6.2 GT correction

Knowing that a step ideally occurs at the highest local amplitude of the accelerometer signal, which is when the foot touches the ground, and also considering the user's reaction time, the GT is firstly shifted to the nearest local minima or maxima, as shown in Figure 3.11.

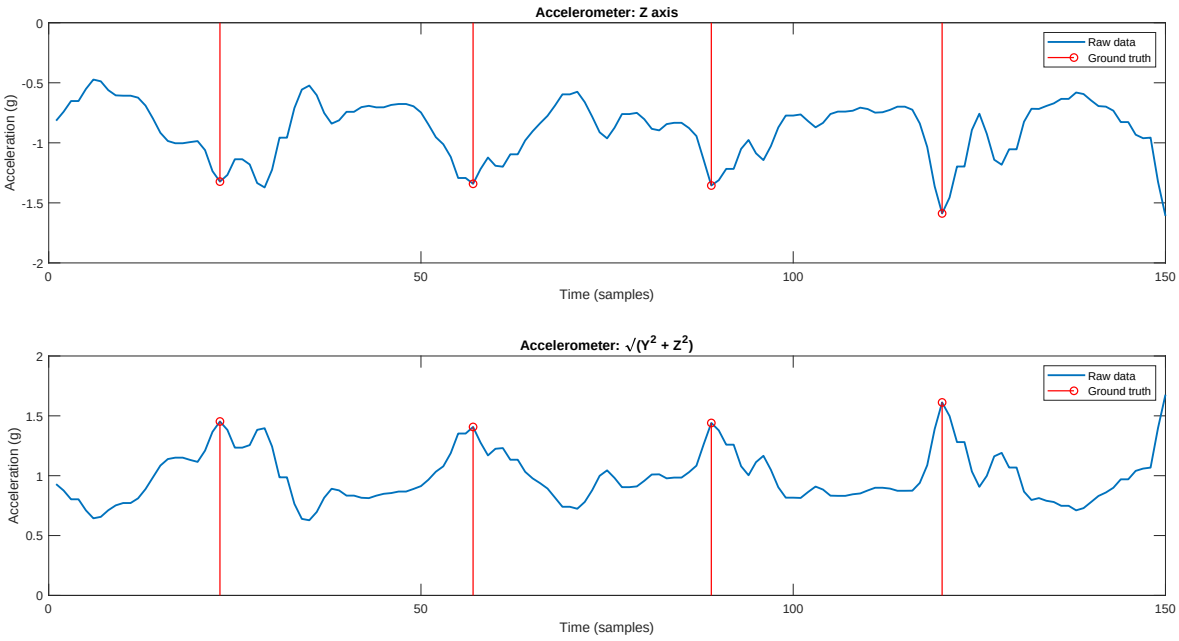


Figure 3.11: Corrected GT, matching accelerometer data (MATLAB)

The MATLAB code for this correction can be found in Appendix B, section B.1.

3.6.3 Low-pass filtering

Due to the slow nature of step events, a second order (avoiding larger delays in higher order filters, due to the need for a real-time response) low-pass Butterworth filter with a cutoff frequency of 5 Hz was applied to both signals, yielding the results in Figure 3.12.

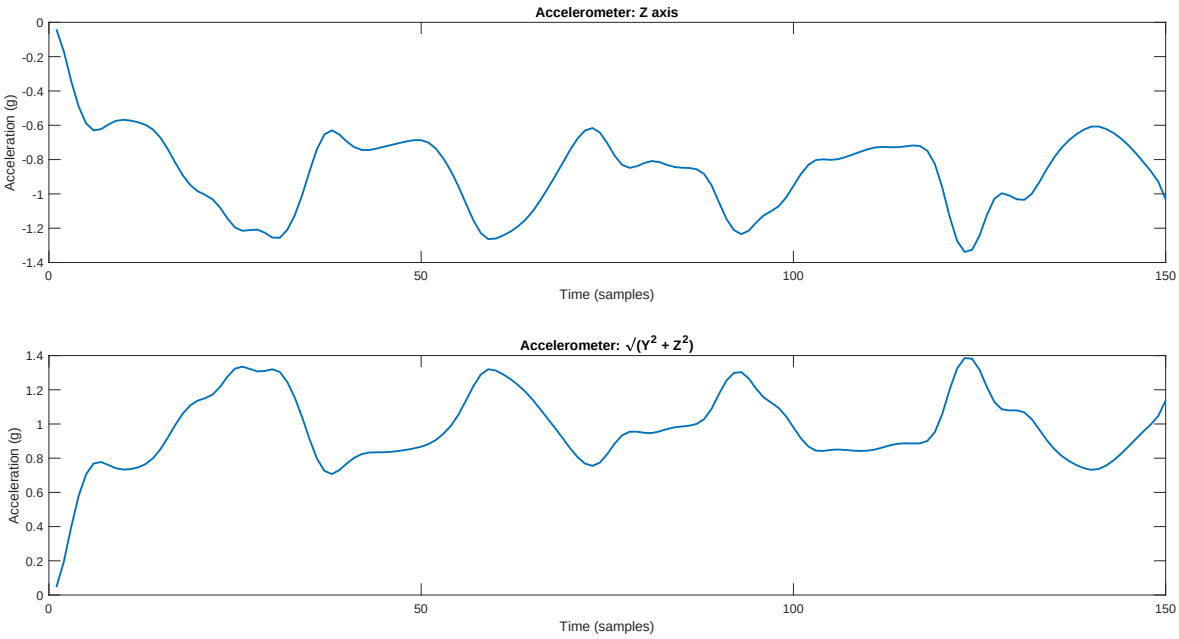


Figure 3.12: Filtered accelerometer data (MATLAB)

3.6.4 Training and testing sets

A total of 33200 samples (1000 steps) were recorded by a single user, randomly chosen from those mentioned in section 3.5, using a Samsung Galaxy A80 smartphone. The data was split into training (26629 samples, 800 steps) and testing (6571 samples, 200 steps) sets.

3.6.5 Detection algorithm

Through an analysis of the filtered signal, the following detection algorithm is proposed:

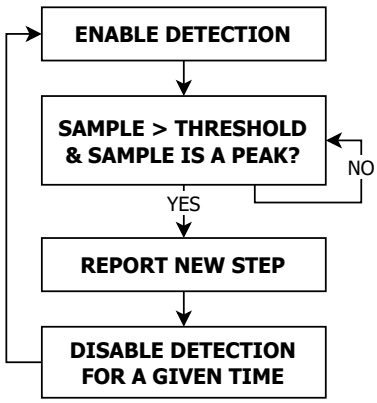


Figure 3.13: Proposed rule-based step detection algorithm

The amount of time during which step detection is disabled should be as large as possible, considering a maximum value of SPM; as the app is not meant to be used while running, a limit of 3 steps per second (180 SPM) is considered, corresponding to a speedy walking pace.

3.6.6 Classifier performance metrics

Before creating an algorithm to refine the threshold that best fits the training data, it is mandatory to understand the difference between accuracy, precision, recall and F1-score.

Accuracy:

An intuitive performance metric, accuracy is the ratio of correct predictions to the total observations. However, if the data set is not balanced (similar number of positive and negative examples), this metric may be misleading, since correctly predicting all members of the predominant class and none of the other class can still result in a high accuracy.

Precision:

A metric inversely related to the rate of false positives, precision (P) is the ratio of true positive predictions (TP) to the total positive predictions, TP and false positives (FP).

$$P = \frac{TP}{TP + FP} \tag{3.1}$$

Recall:

Also known as true positive rate, recall (R) is the ratio of true positive predictions (TP) to the total positive examples observed in the data set, TP and false negatives (FN).

$$R = \frac{TP}{TP + FN} \tag{3.2}$$

F1-score:

Appropriate for unbalanced data sets, the F1-score ($F1$) is a figure of merit combining precision (P) and recall (R), meaning it takes false positives and false negatives into account.

$$F1 = 2 \times \frac{P \times R}{P + R} \tag{3.3}$$

3.6.7 Threshold refinement and attained results

Since the training set is highly unbalanced (800 ones for 25829 zeros), the F1-score will be the metric used to find the appropriate threshold. Additional attention is required regarding correct step detection, which may happen slightly before or after the ones in the GT vector.

Thus, for evaluating the performance of this rule-based classifier, every time a positive prediction is made, a safe window of approximately 133 ms (half the average human reaction time) is considered to verify the GT vector for nearby steps (see Appendix B, section B.2).

Starting with a threshold value equal to the mean of the accelerometer signal, increased in small steps up (down) to the global maximum (minimum) of the $\sqrt{Y^2 + Z^2}$ (Z) accelerometer signals, discarding non-peaks and setting a 180 SPM limit, the following results were obtained:

		Predicted	
	$\sqrt{Y^2 + Z^2}$ Axes	Positive	Negative
GT	Positive	198	2
	Negative	2	6369

Opt. threshold (g)	0.9988
F1-score (%)	99

		Predicted	
	Z Axis	Positive	Negative
GT	Positive	198	2
	Negative	2	6369

Opt. threshold (g)	-0.9489
F1-score (%)	99

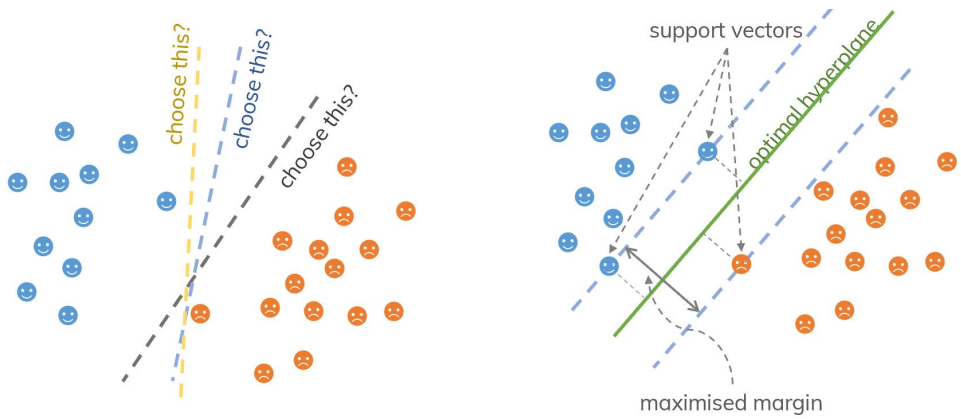
Table 3.1: Confusion matrices for the rule-based classifier

3.7 Machine Learning approach

3.7.1 Support Vector Machine (SVM)

SVM is a supervised learning algorithm that can be used for classification (SVC) and regression (SVR) problems. It is commonly used for smaller datasets, being computationally expensive to process. Again, this particular application is focused on classification.

In an SVM, each data item is represented as a point in the m-dimensional space (where m is the number of features), with each feature being the value of a particular coordinate; classification is achieved by finding the hyper-plane that better separates the different classes.



Source: `https://dinhanhthi.com/img/post/ML/support-vector-machine/svm-1.jpg`

Figure 3.14: Visual representation of the SVM algorithm

As can be seen in Figure 3.14, the best hyper-plane is achieved through a considerate balance between finding the largest margin while still attaining correct classification.

The equation for the m-dimensional hyper-plane is given as:

$$y = w_0 + \sum_{n=1}^{m} w_i x_i = b + w^T X \tag{3.4}$$

where b is the bias term, w^T contains vectors w_1, w_2 ... w_m and X is the feature matrix.

Soft margin vs. hard margin:

If the data is linearly separable and there is no tolerance for misclassifications, using a hard margin is recommended. However, a soft margin is more adequate when a linear boundary is not enough or in applications where the goal is to achieve generality.

This describes the performance of the model on the training and testing sets: a large margin effectively corresponds to more regularization of the SVM weights, preventing over-fitting in the training phase. Therefore, the right margin should be chosen via cross-validation, since it helps to generalize the predictions and allows for better performance on the test data.

Regularized cost function:

In an ordinary (binary class) SVM, the cost function (a measure of how wrong the model can estimate the relationship between its inputs and outputs) is given by:

$$\min_{\theta} C \sum_{i=1}^{m} [y^{(i)} cost_1(\theta^T x^{(i)}) + (1 - y^{(i)}) cost_0(\theta^T x^{(i)})] + \frac{1}{2} \sum_{i=1}^{n} \theta_j^2 \tag{3.5}$$

For a large margin SVM, we have:

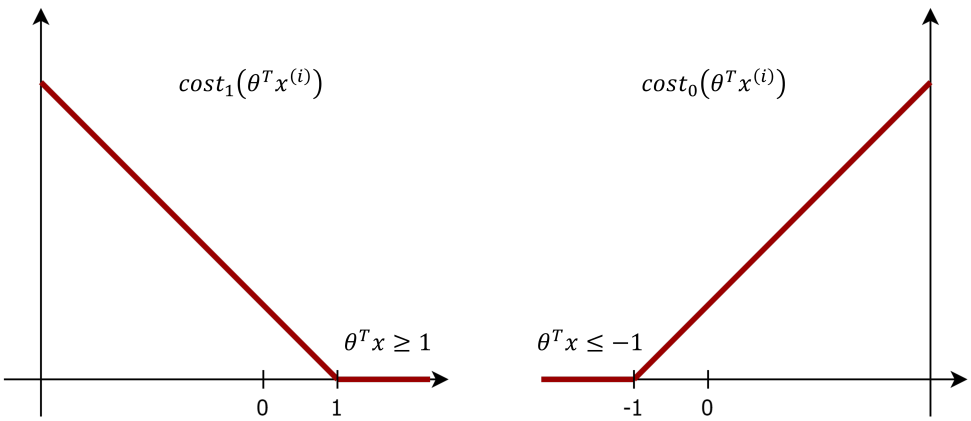


Figure 3.15: Large margin cost functions for $y = 1$ and $y = 0$

SVM parameters:

C is a parameter used to control the error. If C is small (higher bias, lower variance), the penalty for misclassified points is low, and a decision boundary with a large margin is chosen. If C is large (lower bias, higher variance), the SVM tries to minimize the number of misclassified examples to a high penalty, which results in reducing the decision boundary to a small margin - probably resulting in over-fitting. On the other hand, *gamma* (γ), which does not affect the linear kernel, controls the distance of influence of a single training point.

Low values of *gamma* indicate a large similarity radius, which results in more points being grouped together; inversely, higher values only consider points very close to the margin.

Kernel trick for non-linearity and higher dimensions:

SVM algorithms use a set of mathematical functions that are defined as kernels. The function of a kernel is to take data as input and transform it into the required form, by returning the inner product between two given points in a suitable feature space.

In other words, the objective of the kernel trick is to simplify the computation of the dot product between two vectors. Thus, by defining a notion of similarity, the required computation is drastically reduced, even for high-dimensional spaces; the kernel plays a vital role in classification and is also used to analyze certain patterns in a given data set.

In this work, the Gaussian RBF, polynomial, sigmoid and linear kernels are approached; polynomial kernels (Equation 3.6) are commonly used to find the similarity of vectors in a feature space over polynomial original values, making them suitable for non-linear models.

$$k(X, Y) = (a + X^T Y)^b \tag{3.6}$$

The most commonly used kernel is the Gaussian radial basis function (RBF), where the value of the function depends only on the distance from a defined point (Equation 3.7).

$$k(X, Y) = \mathrm{e}^{\gamma \|X-Y\|^2} \tag{3.7}$$

The sigmoid kernel (Equation 3.8) behaves like a two-layer perceptron neural network; there are 2 adjustable parameters in this kernel: the intercept contact c and the slope α.

$$k(X, Y) = tanh(\alpha X^T Y + a) \tag{3.8}$$

Lastly, the linear kernel (Equation 3.9) is the simplest, resulting from the addition of the inner product of two vectors to a constant.

$$k(X, Y) = a + X^T Y \tag{3.9}$$

3.7.2 Implementation

As in subsection 3.6.4, the data set consisting of 1000 steps in 33200 total samples is here divided into a training and cross-validation set (80%) and a testing set (20%).

The SVM classifier for step detection, the cross-validation (CV) method and the performance metrics were all implemented using the Python *sklearn* library in Jupyter Notebook.

Feature windows and SVM kernel selection:

Individual samples were firstly grouped in windows of samples, so that in a certain way the model could develop a step paradigm and, therefore, identify step and non-step occurrences.

Since the main goal of the present work is to implement real-time step detection, such detection should happen in a reasonably short time after the foot hits the ground. So, the feature windows are formed around values in the ground truth vector, with variable sizes to the left (past) and to the right (future) of these points. Obviously, 'future' denotes a small delay, a number of samples to the right that does not imply huge latency (hence real-time).

These variable sizes, referred to as left and right window sizes, vary according to the following constraints, based on the relevance of past events and the human reaction time:

- less than 0.5s for the left window size;
- a maximum of 50ms for the right window size.

At 60 Hz, the maximum left and right window sizes are of 29 and 3 samples, respectively.

Within a *while* loop, every possible combination of left and right window sizes were evaluated, as well as the performance of the four previously mentioned SVM kernels for each combination, with default settings. At each iteration, some of the resulting feature windows were discarded so that an equal number (800) of steps and non-steps form a balanced set.

The performance of the four different SVM models was evaluated at each iteration through a 5-fold CV method, so as not to over-fit the training data. Since the training and CV set is now balanced, accuracy is the performance metric used for determining the best kernel and the best left and right window sizes for grouping the data samples.

The best obtained results are shown in the table below, using the modulus of the accelerometer - $\sqrt{Y^2 + Z^2}$ - which performed better than just the Z axis (94.25%).

Kernel	**Left (samples)**	**Right (samples)**	**Acc. (%)**
Polynomial	5	2	94.75

Table 3.2: Best performing kernel and window sizes

An example of a step and a non-step event, with the best window sizes, is shown below:

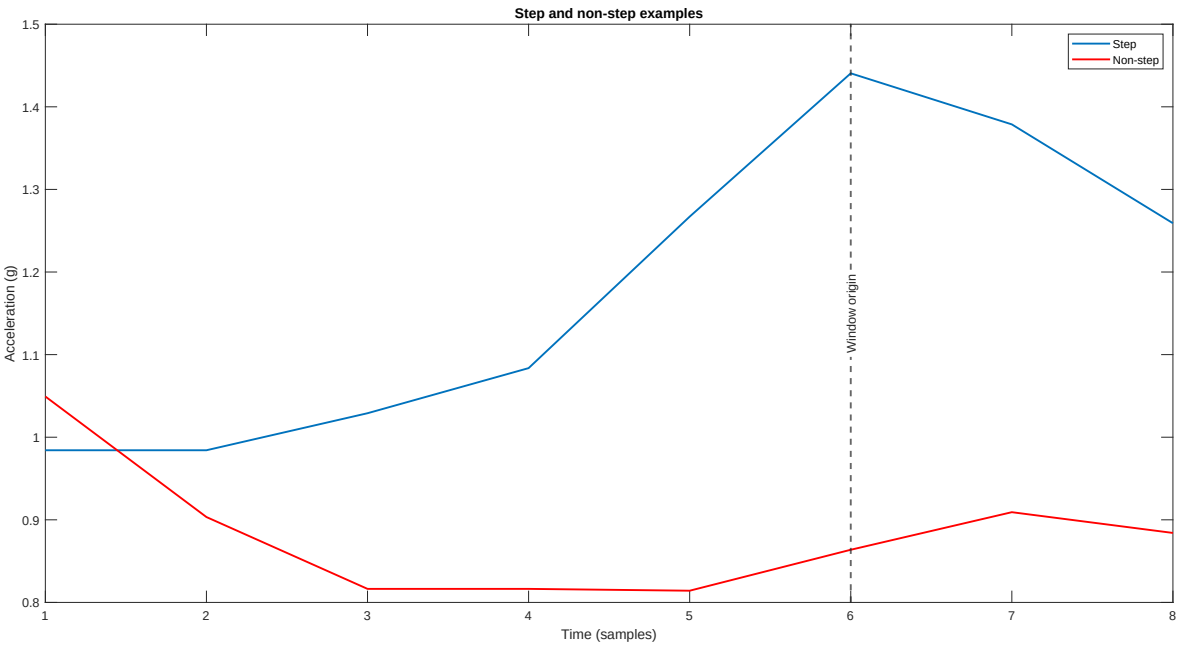


Figure 3.16: Step and non-step examples

Figure 3.17 shows the evolution of the CV error with the left window size.

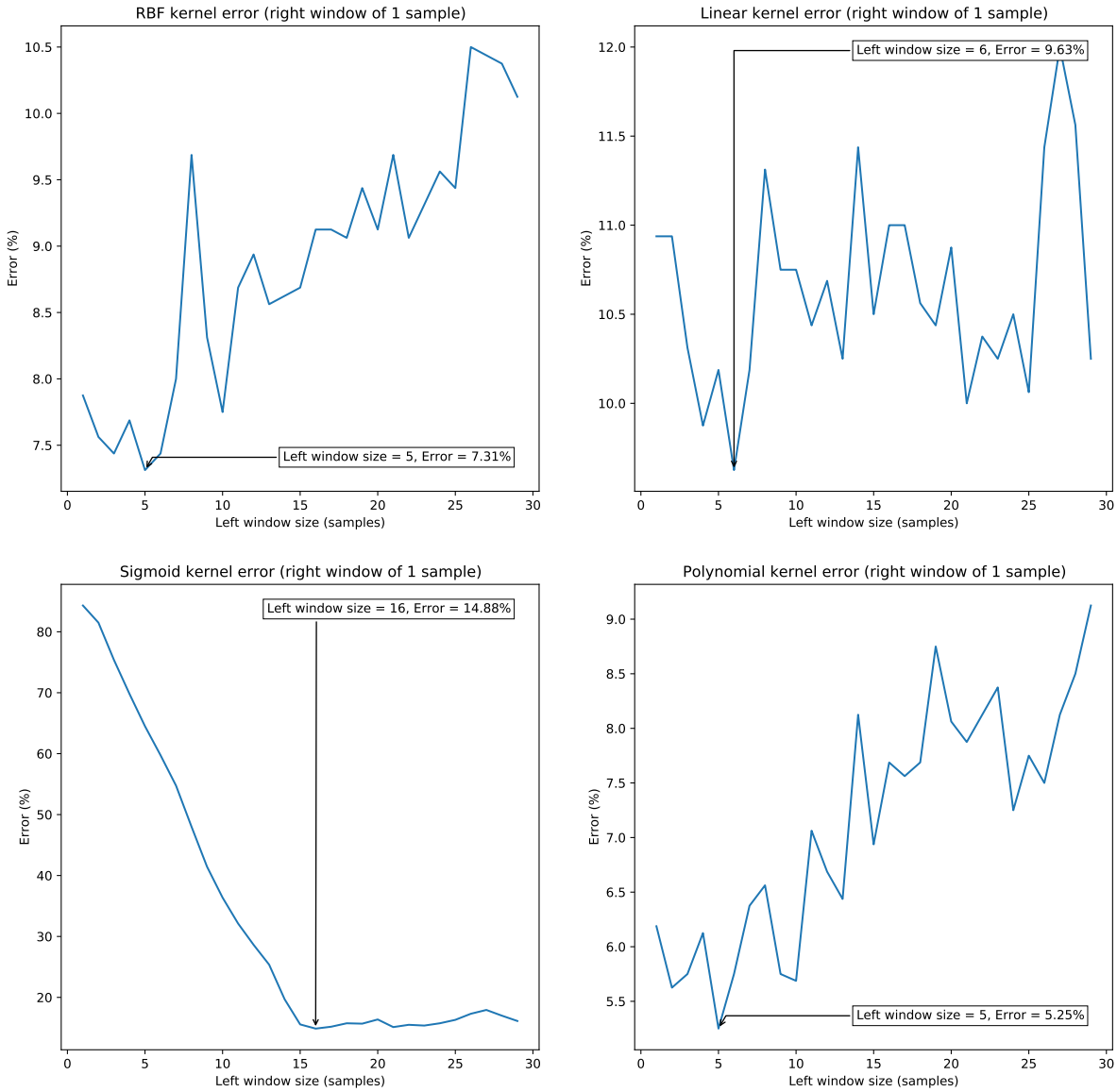


Figure 3.17: CV error evolution for the four SVM kernels

Hyper-parameter selection

Considering the previous results, a separate Jupyter Notebook script was created to refine the hyper-parameters and the degree of the polynomial kernel SVM that would best perform on a 5-fold CV set. The refinement iterations yielded:

Iter.	***C***	***gamma***	***degree***	**Candidates**	**Fits**	**Score (%)**
1	0.85	0.9	4	245	1225	95.062
2	0.81	0.87	4	81	405	95.125
3	0.5	0.85	4	35	175	95.062

Table 3.3: Refinement of SVM degree and parameters

3.7.3 Performance on testing set

After parameter refinement, the model was evaluated by its performance on the testing set. As such, some additional constraints regarding the operation of the smartphone application were taken into account, as already mentioned for the rule-based method:

- after a step is detected by the model, the system becomes idle for a third of a second (no subsequent steps are expected in such a small time interval);
- thus, since no consecutive sample detections are possible, positive model predictions are considered to be valid in a confidence interval of a sixth of a second (before and after the positive prediction); in other words, since the GT vector only indicates a '1' at a specific instant, as soon as the model predicts a step the GT vector is evaluated in the range of the confidence interval, validating the detection if a step truly occurred in that interval, and resulting in a false positive if no occurrences were found.

Having this in consideration, the samples of the original testing data were grouped into windows of the previously mentioned sizes, so that the model could predict step occurrences.

An F1-score of 95.84% was attained, corresponding to the following confusion matrix:

		Predicted	
		Positive	Negative
Real	Positive	196	4
	Negative	13	6351

Table 3.4: Confusion matrix for the SVM classifier

3.8 Conclusions

The decision boundary of the SVM model could not be graphically represented since the feature window is 8-dimensional; this would have been a very interesting representation of the features. In comparison to the presented state-of-the-art solutions, this model performed very well for such a simple, real-time implementation. However, due to the much higher performance of the simple rule-based algorithm, this will be the choice for application development.

Chapter 4

Application development

4.1 *Modus operandi*

4.1.1 Extracting tempo information from walking

Having in mind that the collected step information will be used to define the music playback tempo, a balance between a quick response and immunity to falsely detected steps should be achieved. For this, the following functionalities were promptly established:

- BPM is calculated from a weighted average of the last 5 steps (immunity to false steps);
- the interval between the last 2 steps is the most determinant factor (quick response).

Figure 4.1 illustrates the intended tempo extraction functionality.

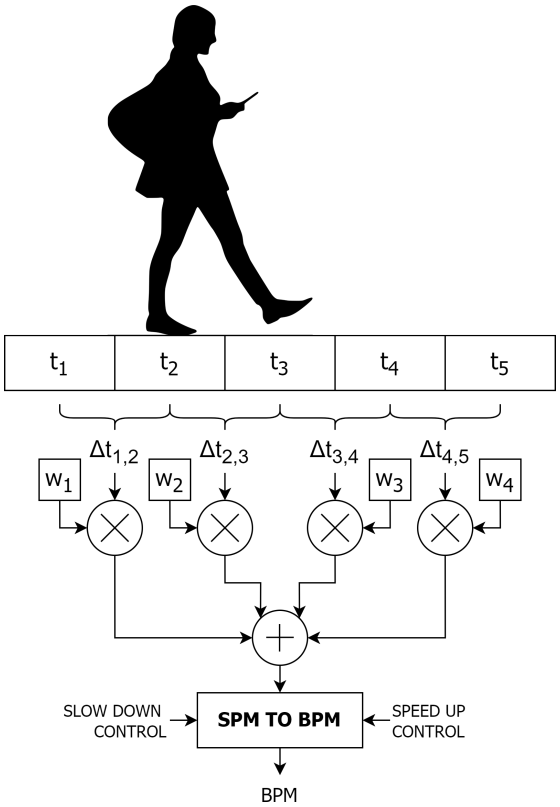


Figure 4.1: Extracting playback tempo from the user's walking pace

4.1.2 Music playback

The main goal of this development is to attain the ability to import, parse and play MIDI files (discussed in the following sections) at the constantly updated walking tempo. As soon as playback is activated, the user must provide 5 initial steps to determine the starting tempo.

A simple block diagram of the intended mechanism is shown in Figure 4.2.

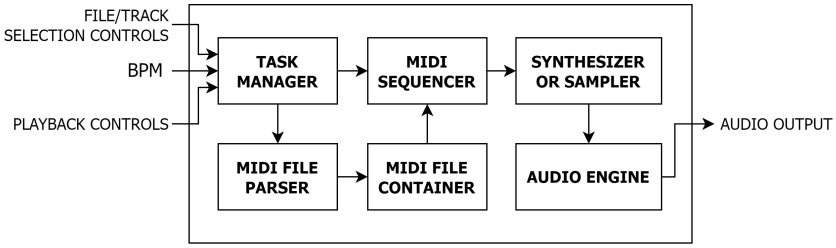


Figure 4.2: Simple depiction of the internal music playback mechanism

4.1.3 Achieving entrainment

As discussed in Chapter 2, helping users achieve entrainment is one of the core features of 'Practice As You Walk'. A particularly interesting detail is that not only does the smartphone adapt to the user's pace, but the user also adapts to the rhythm of the music being played.

Possibly, the leading challenge while using this application is understanding the presence of these two feedback paths (user-to-device and device-to-user), so as to be able to achieve a state of synchronization, which indicates that the system, as a whole, is balanced.

4.2 Initial prototype

An early draft of the application was developed with the sole purpose of demonstrating its functionality; at this point, the implementation was comprised of the following:

- modulus of the triaxial accelerometer signal (agnostic to phone position);
- 5 Hz, second-order low-pass filtering of the accelerometer signal;
- weighted average of the last 5 steps for tempo extraction;
- support for a single musical phrase, stored in memory as a sequence of numbers;
- each number corresponded to a certain sample of a piano note, stored as a .wav file;
- initial tempo is provided by the user, through 5 initial steps before playback.

This draft was very important for understanding the flow within the Unity platform, especially regarding the manipulation of *AudioSource* components; in terms of performance, no objective data was collected, but every person involved in this preliminary phase saw potential for 'Practice As You Walk' and found attaining entrainment to be fun and rewarding.

Author credits:

The Butterworth low-pass filter is part of the 'Unity-IIR-Realtime-Filtering' class made available by Marius Rubo on GitHub [38], which provides real-time filtering of streaming data.

4.3 Brief overview of the MIDI standard

4.3.1 Introduction

The Musical Instrument Digital Interface (MIDI) is a data communication protocol that specifies a means to exchange information between music systems and related equipment.

MIDI is not a musical language and does not directly describe musical sounds [39].

4.3.2 Hardware specification

The MIDI protocol specifies conditions for the data format, regarding the data messages that one device transmits to another, and for the hardware interface, the physical connection between those devices; this interface is out of the scope of this work and will not be approached.

4.3.3 MIDI messages

Every message denotes a musical event, consisting of one or more bytes of digital data, and each byte can be either a status byte (most significant bit is '1') or a data byte (most significant bit is '0') [39]. For instance, Figure 4.3 shows a *Note On* message.

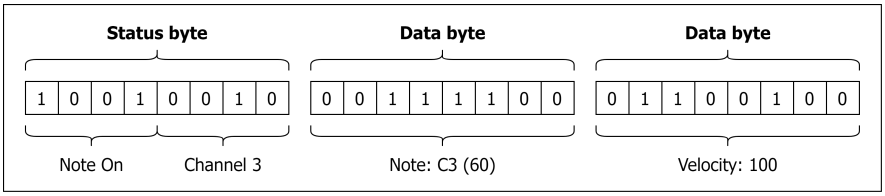


Figure 4.3: MIDI *Note On* message

There are two main categories of MIDI messages: *Channel* and *System* messages. The first category typically controls aspects of performance, such as *Note On/Off*, while *System* messages control the full MIDI system, such as *System Exclusive (SysEx)* messages [39].

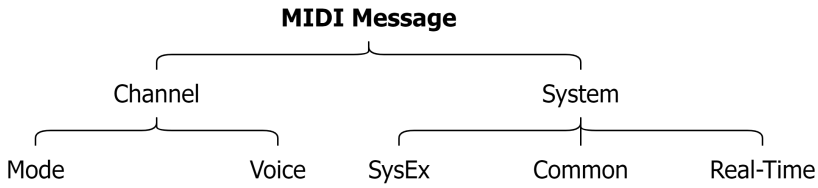


Figure 4.4: Categories of MIDI messages

4.3.4 *Running Status*

To increase efficiency, the MIDI specification allows for a message to be sent without its *Status* byte if the previous message had the same *Status* (excluding *Real-Time* messages).

For example, *Control Change (CC)* messages, which affect sounds already initiated by *Note On* messages, are comprised of one *Status* byte and two *Data* bytes. If a three-byte *CC* message is followed by another *CC* message, the latter may only contain the two *Data* bytes.

4.3.5 Standard MIDI File (SMF)

SMFs (.mid) are composed of all the MIDI instructions that must be evaluated for playing a certain musical excerpt, such as note generation, sound selection and volume control.

Structure:

The internal structure of an SMF consists of groups of data known as chunks; each of these chunks is comprised of a 4-character chunk identifier, a 32-bit value containing the length of the chunk (in bytes) and the chunk data itself. The types currently defined in the specification are header chunks ('MThd'), at the beginning of an SMF, and track chunks ('MTrk').

File format, number of track chunks and division are all part of the header data [39].

Formats:

Three different SMF formats are defined by the specification: 0, 1 and 2; though each contain different numbers/types of tracks, the MIDI data is largely the same among them:

- format 0, the simplest, consists of a single multichannel track;
- format 1 consists of multiple tracks that are played simultaneously;
- format 2 provides independent tracks with their own meters and tempos [39].

For now, SMF format 2 is still widely unimplemented, thus is not approached in this work.

Division:

There are two possible interpretations for the division value; within the context of this work, it is the quarter-note timing resolution, or the number of pulses per quarter note (PPQN).

Delta times:

Every track event starts with its delta time (elapsed time since the preceding track event); the number of bytes needed to represent it may vary, being limited to 4 bytes [39].

4.3.6 Additional information

Although brief explanations of fundamental MIDI concepts that are part of the present work were presented, reading the comprehensive 'Official MIDI Specifications' [40] is advised.

4.4 Development in Unity

4.4.1 Third-party libraries

smflite:

A class library for handling SMFs in Unity, by Keijiro Takahashi, is available on GitHub [41] and was incorporated into this project for easier parsing and manipulation of these files.

4.4.2 User interface

A very basic user interface was designed, allowing users to select a MIDI file, the desired track (in the case of format 1 SMFs), the step detection threshold (referred to as sensitivity) and to control the playback mechanism. As shown in Figure 4.5, there is a section displaying the remaining steps before playback starts, which displays the current BPM during playback.

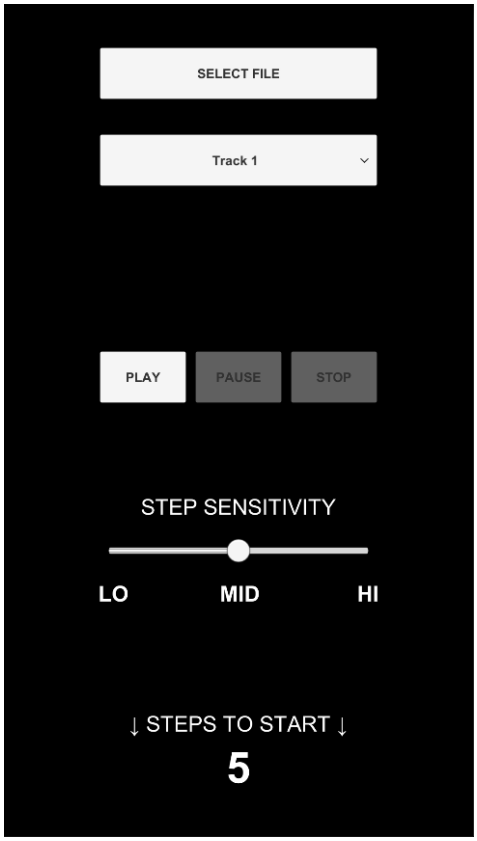


Figure 4.5: User interface of the app developed in Unity

4.4.3 Internal structure and operation

The C♯ scripts which define the behavior of the app were organized in four folders, according to their content: 'General', containing scripts for MIDI file importing, low-pass filtering and management of global variables; 'GameObjects', for scripts which mostly interact with *GameObjects*, such as playback controls, track selection and step sensitivity; 'MIDISequencer', for the operation of the MIDI sequencer; 'SMFLite', for each class of the *smflite* library.

Importing MIDI files:

Using the free *Native File Picker* asset available on the Unity Asset Store, the application prompts the user to select a MIDI file, asking for memory access permission in the first run.

Then, all bytes from the selected file are read into a byte array, from which an *smflite.MidiFileLoader* extracts division (PPQN) and tracks into an *smflite.MidiFileContainer*.

MIDI track selection:

By default, an *smflite.MidiFileSequencer* is created for the first track. However, if the imported file is a format 1 SMF, the user may choose any track in the *Dropdown* shown in Figure 4.5, for which a new *smflite.MidiFileSequencer* is created, replacing the previous one.

MIDI track sequencing:

As soon as the user hits the 'Play' button, step detection (using the algorithm in section 3.6 with a configurable threshold) starts and five initial steps are mandatory to determine the initial playback tempo in BPM (which is 120 by default). The averaging weights are:

$$\Delta T = 0.1 \cdot \Delta t_{1,2} + 0.25 \cdot \Delta t_{2,3} + 0.3 \cdot \Delta t_{3,4} + 0.35 \cdot \Delta t_{4,5} \tag{4.1}$$

where $\Delta t_{4,5}$ is the time interval (in seconds) between the two most recent steps.

The playback speed, in BPM, is then obtained through Equation 4.2.

$$\text{BPM} = \frac{60}{\Delta T} \tag{4.2}$$

At the fifth step, the *smflite.MidiFileSequencer.pulsePerSecond* parameter is updated using *smflite.MidiFileContainer.division* (PPQN) and the BPM value, as in Equation 4.3.

$$pulsePerSecond = \frac{\text{BPM}}{60} \cdot \text{PPQN} = \frac{\text{PPQN}}{\Delta T} \tag{4.3}$$

The *smflite.MidiFileSequencer* then advances to the first *smflite.MidiTrack.DeltaEventPair* which, as the name implies, contains an event and its delta time. From here on, at each frame update (corresponding, for instance, to 0.017 seconds, the inverse of a 60 Hz frame rate), the *smflite.MidiFileSequencer* advances to the delta time that is 0.017 seconds ahead, sending every *smflite.MidiEvent* in between the frames to the audio playback mechanism.

Audio playback:

For this application, 88 audio samples (.wav, 44.1 kHz) sampled from a Roland RD-700 keyboard (all keys) were loaded as *Audio Clips*. Hence, 88 *Audio Sources* were created, each with its corresponding *Audio Clip*. For each *smflite.MidiEvent* received, the system evaluates:

- is it a *Status* + *Data* message?
  - is it a *Note Off* event?
  - is it a *Note On* event?
  - is it a *CC* piano pedal event?
- is it a *Data*-only message (*Running Status*)?
  - is it a *Note Off* event?
  - is it a *Note On* event?
  - is it a *CC* piano pedal event?

Then, the system acts by playing (at the right volume) or stopping the corresponding *Audio Clip*; please see the example code in Appendix A, section A.3 and also refer to [40].

4.4.4 Identified problems

First testing phase:

An immediately identified problem when testing the application was the presence of sudden pops in the playback. This was due to the harsh stopping of *Audio Clips* at *Note Off* events.

Instead, a quick fade out procedure was implemented.

Second testing phase:

Small pops in audio playback were still identified when the same note was pressed at least twice in a short time, a problem that was missed in the first testing phase. It now happens due to the *Note On* event both when the note is already playing or did not finish fading out.

For this, an extra set of 88 *Audio Sources* was added, so that when a note is triggered while it is playing or fading out in the first *Audio Source* there is a chance to play it seamlessly.

Third testing phase:

The previously identified pops have disappeared; however, slight crackles can still be heard in playback. The root cause of this problem was not found, as the project's audio settings were already optimized for performance, as shown in Figure 4.6.

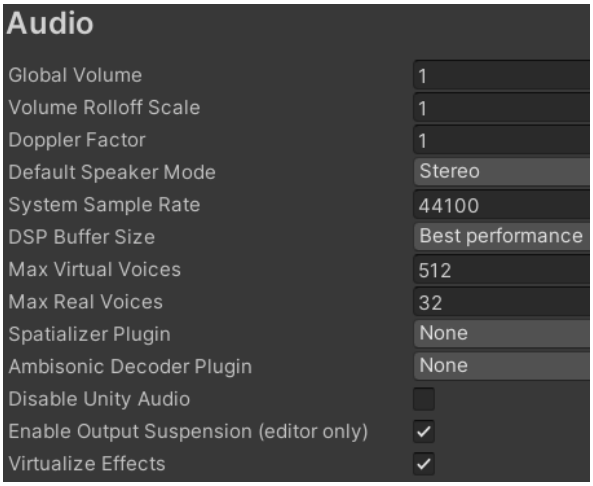


Figure 4.6: Audio settings for the Unity project

4.4.5 Conclusions

While the application performed correctly in terms of retrieving playback speed from the user's pace, importing and parsing MIDI files, sequencing them at the retrieved tempo and translating MIDI events to actual instrument sounds, it lacks glitch-free audio playback.

Regarding the initial requirements for the application, importing a format 0 SMF will load a single track where the MIDI data of all channels is merged (for example, all voices in a choir); importing a format 1 SMF will load multiple tracks, from which the user can choose a single one for playback (for example, a soprano may want to practice along the tenor melody).

4.5 Development in Android Studio

4.5.1 Overview

Based on IntelliJ IDEA, Android Studio is the official IDE for Android app development. Aside from IntelliJ's outstanding code editor and developer tools, Android Studio offers additional features that help streamline app development, such as the following [42]:

- a versatile Gradle-based build system;
- a fast emulator with multiple features;
- code and resource changes to a running app without restarting;
- extensive testing tools and frameworks.

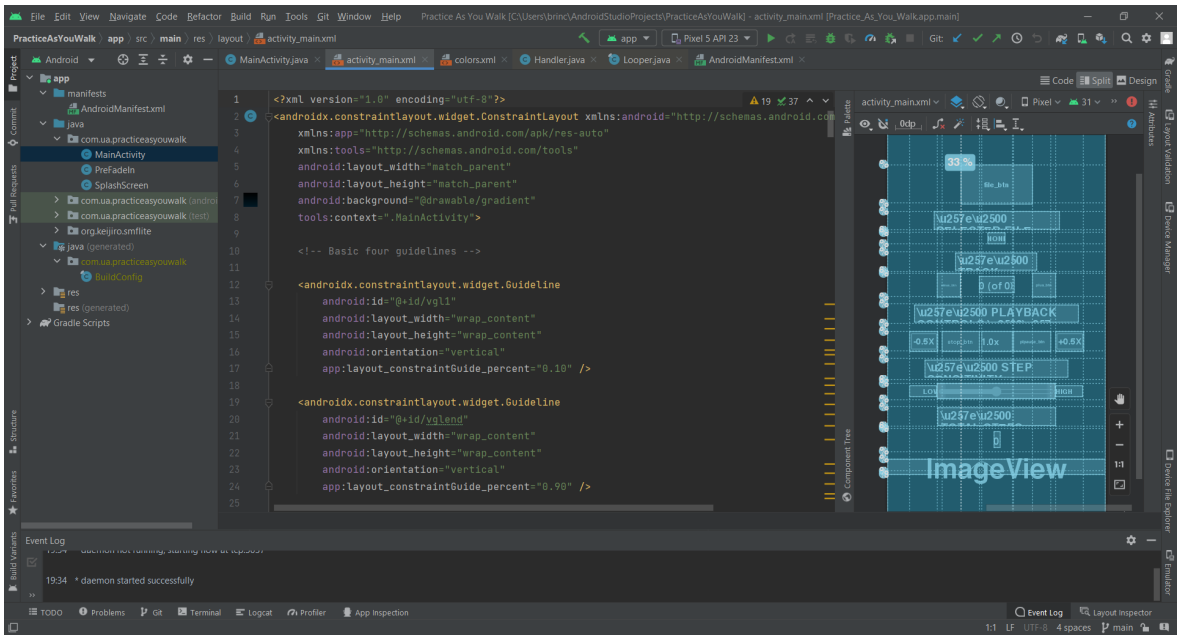


Figure 4.7: Typical layout of Android Studio

4.5.2 Third-party libraries

Midi Driver:

This Android MIDI driver using the Sonivox EAS library [43], developed by Bill Farmer, was the driving force behind moving to Android Studio. It presents an immediate solution to the audio glitches found in the Unity app, through the use of a fully functional open source MIDI synthesizer library, thus supporting multiple instrument sounds.

iirj:

The application developed in Android Studio employs a Butterworth low-pass filter that is part of the 'IIR1' library, made available by Bernd Porr on GitHub [44], which provides sample by sample filtering for realtime processing of incoming accelerometer data.

4.5.3 Retaining functionalities from the Unity app

The two official languages for Android programming are Java and Kotlin; as the third-party MIDI driver is written in Java, it is more practical to develop this app in the same language.

The first step was to convert the *smflite* library from C♯ to Java, so that the entire functionality of the Unity app regarding MIDI is preserved (see Appendix C, section C.1).

4.5.4 User interface

Using XML for designing the user interface allowed for better control over its various elements, resulting in a less boring, easy-to-use and modern-looking application:

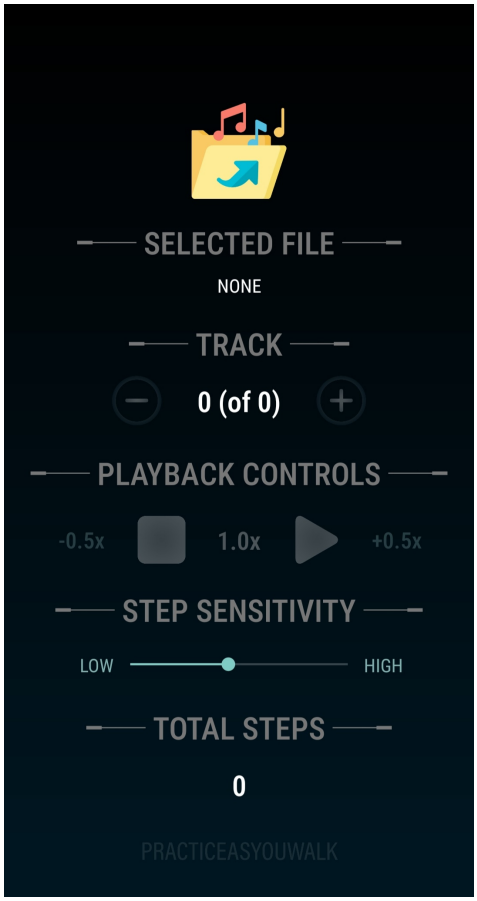


Figure 4.8: User interface of the app developed in Android Studio

4.5.5 Main implementation differences

The most important advantage of this implementation is the complete abstraction from the audio playback mechanism. Specifically, all the code in Appendix A, section A.3 is replaced by sending all the three bytes from the MIDI messages to the third-party MIDI driver.

Additionally, the current BPM is no longer shown during playback so as not to distract the user from the naturally occurring entrainment, and the playback speed controls missing in the Unity app were added to offer flexibility (e.g. playback BPM can be twice the SPM).

This page intentionally left blank.

Chapter 5

Validation and discussion

5.1 Final version of the Android application

With a very straightforward user interface, requiring no user manual, the application only requires a MIDI file to be imported. Then, the track selection controls are activated, as well as the playback controls. It is then up to the user to decide which track to play, what multiplier to apply to their walking pace and what step detection sensitivity to configure.

Similarly to the application developed in Unity, the user should provide five initial steps after the 'Play' button is pressed for establishing an initial playback tempo.

Screenshots of the splash screen and the app during execution are shown in Figure 5.1.

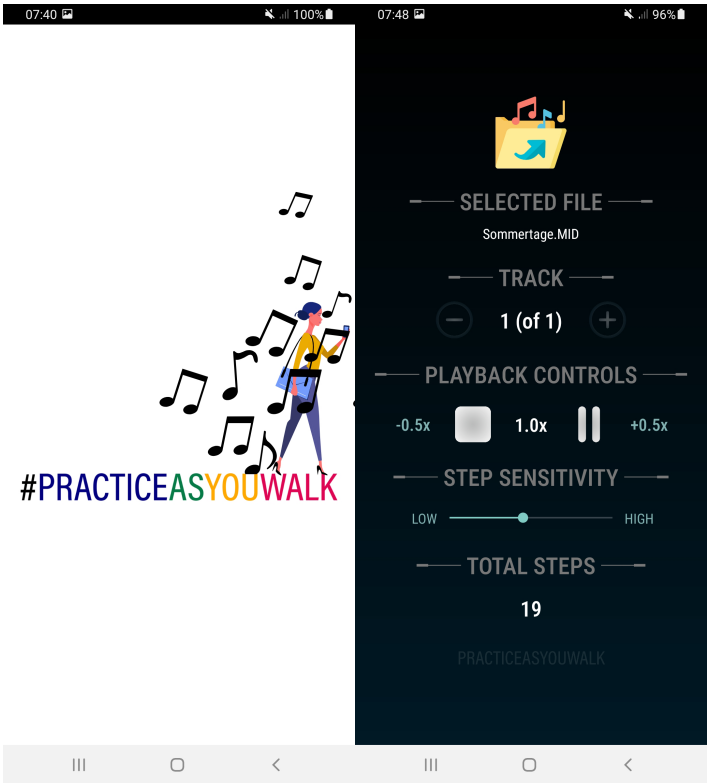


Figure 5.1: Application: splash screen (left) and interface (right)

5.2 Validation

5.2.1 Test group

'Practice As You Walk' was presented to a youth choir, 'Canarinhos de Itabirito', from ages eleven to eighteen, from Itabirito, Minas Gerais (Brazil), which is directed by the previously mentioned collaborator (Lana, É.). Twenty-four participants integrated the test group.

5.2.2 Test procedure

All participants were handed the Android application installation file (.apk) and installation instructions. Through an online meeting, the choir director was guided through the development of the application and the need to export format 0 SMFs, containing a single track, to play all voices at once, or format 1 SMFs, containing multiple tracks, to reproduce a single (selectable) track at once. The choir members were given two SMFs, one of each format, and practiced using both files in the application, at their own pace, for a month.

5.2.3 Questionnaire

Aiming to understand the effects of the application in the participants' practice routine, a questionnaire was created and made available via Google Forms, regarding the topics below:

- **1)** age and voice type;
- **2)** aesthetic appearance of the user interface;
- **3)** usability of the playback controls;
- **4)** performance of the step detection algorithm;
- **5)** effectiveness of the sensitivity adjustment slider;
- **6)** interestingness of this type of musical practice;
- **7)** innovativeness of the application;
- **8)** usefulness for learning hard musical excerpts;
- **9)** improvements on the experience as a choir member;
- **10)** possibility of negative effects in the learning process;
- **11)** general satisfaction with the application;
- **12)** desire for future application developments and releases;
- **13)** willingness to download the application if published online;
- **14)** opinions and suggestions.

5.2.4 Results

With an average age of 15 years, the participants' voice types were relatively well distributed; all individuals found this type of musical practice to be interesting. The complete results (in Portuguese) can be found in Appendix D. The average scores for applicable topics were:

- **2)** 3.67 out of 5;
- **3)** 4.13 out of 5;
- **4)** 3.79 out of 5;

- **5)** 4,13 out of 5;
- **6)** 4.58 out of 5;
- **11)** 4.25 out of 5.

5.2.5 Future work

The results from the questionnaire were discussed in a subsequent online meeting with all members of the choir, in which some valuable additional feedback was gathered.

Improvements:

One participant astutely reported that it was easier to move from a slow tempo to a fast tempo than the other way around. This must be investigated, and can be due to the weights attributed to the intervals between steps in the tempo extraction procedure.

Additionally, considering the average results in the performance of the step detection algorithm, it might have happened that the high F1-score previously reported is highly dependant on the performance of the accelerometer, or there might be some faults in the code that need to be analyzed and corrected, but the algorithm should achieve generality. Results regarding the user interface also show the need for a more user-friendly design. An essential improvement would be the ability to simultaneously play all tracks in a format 1 SMF.

Additional functionalities:

Two participants suggested that the application should support MP3 files (see Appendix D). Rather than a matter of supporting different formats, this application currently deals with music performance information and adjusts the rate at which this information is sent to the audio playback mechanism. However, it would be very interesting to incorporate a time-stretching functionality in the future, similar to the mechanism referred in [27].

Another participant suggested changing music dynamics depending on the force with which the foot hits the floor, or by lowering or raising the smartphone. While also a fantastic idea for the evolution of 'Practice As You Walk', this should require many additional considerations.

Publishing the application:

As something that was considered since the first prototype was developed, publishing the application may be a reality if it matures enough in comparison to the current results, especially since most participants would be eager to download it from an online platform.

This page intentionally left blank.

Appendix A

C♯ code

Note: C♯ scripts in Unity inherit variables and methods from the MonoBehaviour class; therefore, the following inherited methods, used within this project, behave as described:

- *Start():* called before the first frame update, used mainly for initializing variables;
- *Update():* the core of the application, being called once every frame update;
- *OnApplicationQuit():* for ending ongoing tasks or writing logfiles when exiting the app.

A.1 Reading gyroscope and accelerometer sensors

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.InputSystem;
using System.Text;
using System.IO;

public class PrelimTest : MonoBehaviour
{
    int             i;
    string          path;
    List<double>    time_arr;
    List<Vector3>   acc_arr, gyr_arr;
    TextWriter      time_w, acc_w, gyr_w;

    // Start is called before the first frame update
    void Start()
    {
        time_arr = new List<double>();
        acc_arr  = new List<Vector3>();
        gyr_arr  = new List<Vector3>();

        i = 0;
        path = "/storage/emulated/0/Android/data/com.developer.app/logs/";

        if (!Directory.Exists(path))
        {
            Directory.CreateDirectory(path);
        }

        time_w = new StreamWriter(path + "Timestamp.txt", false, Encoding.UTF8);
        acc_w  = new StreamWriter(path + "AccData.txt", false, Encoding.UTF8);
        gyr_w  = new StreamWriter(path + "GyroscopeData.txt", false, Encoding.UTF8);

        InputSystem.EnableDevice(Accelerometer.current);
```

```
        Input.gyro.enabled = true;
    }

    // Update is called once per frame
    void Update()
    {
        acc_arr.Add(Input.acceleration);
        gyr_arr.Add(Input.gyro.rotationRate);
        time_arr.Add(Time.realtimeSinceStartup);
    }

    private void OnApplicationQuit()
    {
        for (i = 0; i < time_arr.Count; i++) // all lists have the same size
        {
            time_w.WriteLine(time_arr[i]);
            acc_w.WriteLine(acc_arr[i].x + ";" + acc_arr[i].y + ";" + acc_arr[i].z);
            gyr_w.WriteLine(gyr_arr[i].x + ";" + gyr_arr[i].y + ";" + gyr_arr[i].z);
        }
        acc_w.Close();
        gyr_w.Close();
        time_w.Close();
    }
}
```

A.2 Adding step/non-step labels (expands section A.1)

1) Additional set of classes in use:

```
using UnityEngine.UI;
```

2) Declaration of additional variables to control GameObjects:

```
public class PrelimTest : MonoBehaviour
{
    int         totalSteps;
    List<int>   step_arr;
    Text        myText, myText2;
    public      Button myBtn, myBtn2;
    TextWriter  step_w;
```

3) Initialization of new variables and assignment of callback functions

```
    void Start()
    {
        totalSteps = 0;
        step_arr   = new List<int>();

        step_w  = new StreamWriter(path + "StepsOrNot.txt", false, Encoding.UTF8);

        myBtn   = GameObject.Find("Canvas/Button").GetComponent<Button>();
        myBtn2  = GameObject.Find("Canvas/PressBtn").GetComponent<Button>();
        myBtn.onClick.AddListener(TaskOnClick);
        myBtn2.onClick.AddListener(TaskOnClick2);

        myText  = GameObject.Find("Canvas/Text").GetComponent<Text>();
        myText2 = GameObject.Find("Canvas/TextNotes").GetComponent<Text>();
        myText.text  = "RECORDING SENSOR DATA...";
        myText2.text = "0";
```

4) Updating new variables at each frame

```
    void Update()
    {
        step_arr.Add(0); // may change to '1' if TaskOnClick2() is called
        myText2.text = totalSteps.ToString();
```

5) Definition of callback functions

```
// 'Save & exit' button
void TaskOnClick()
{
    Application.Quit();
}

// 'Add step' button
void TaskOnClick2()
{
    if (step_arr.Count == 0)
    {
        step_arr.Add(1);
    }
    else
    {
        // replace '0' added in Update() with '1'
        step_arr[step_arr.Count - 1] = 1;
    }
    totalSteps++;
}
```

5) Additional tasks when closing the application:

```
private void OnApplicationQuit()
{
    for (i = 0; i < time_arr.Count; i++) // all lists have the same size
    {
        step_w.WriteLine(step_arr[i]);
    }
    step_w.Close();
```

A.3 Example code for interpreting MIDI messages

```
void ApplyMessages(List<MidiEvent> messages)
{
    if (messages != null)
    {
        foreach (var m in messages)
        {

            // Status + data message
            if ((m.status & 0x80) == 0x80)
            {
                // NOTE OFF EVENT (should only impact the playback system IF pedal is OFF)
                if ((m.status & 0xF0) == 0x80)
                {
                    currStatus = 0;
                    // add code to stop this note: (m.data1 & 0x7F)
                }

                // NOTE ON EVENT
                else if ((m.status & 0xF0) == 0x90)
                {
                    currStatus = 1;
                    currVolume = (m.data2 & 0x7F) / 127.0f;
                    if (currVolume == 0) // NOTE ON WITH 0 VELOCITY (NOTE OFF, should only impact
                        the playback system IF pedal is OFF)
                    {
                        // add code to stop this note: (m.data1 & 0x7F)
                    }
                    else
                    {
                        // add code to play this note: (m.data1 & 0x7F)
                    }
```

```
                }

                // CC EVENT
                else if ((m.status & 0xF0) == 0xB0)
                {
                    currStatus = 2;
                    if ((m.data1 & 0x7F) == 0x40) // PEDAL EVENT
                    {
                        if ((m.data2 & 0x7F) == 0)
                        {
                            pedalOn = false;
                        }
                        else
                        {
                            pedalOn = true;
                        }
                    }
                }

                // OTHER UNUSED COMMANDS
                else
                {
                    currStatus = -1;
                }

            }

            // Data-only message (running status)
            else if ((m.status & 0x80) == 0x00)
            {
                if (currStatus == 0) // Running status: NOTE OFF (should only impact the playback
                     system IF pedal is OFF)
                {
                    // add code to stop this note: (m.data1 & 0x7F)
                }
                else if (currStatus == 1) // Running status: NOTE ON
                {
                    currVolume = (m.data2 & 0x7F) / 127.0f;
                    if (currVolume == 0) // NOTE ON WITH 0 VELOCITY (NOTE OFF, should only impact
                         the playback system IF pedal is OFF)
                    {
                        // add code to stop this note: (m.data1 & 0x7F)
                    }
                    else
                    {
                        // add code to play this note: (m.data1 & 0x7F)
                    }
                }
                else if (currStatus == 2) // Running status: CC EVENT
                {
                    if ((m.data1 & 0x7F) == 0x40) // PEDAL EVENT
                    {
                        if ((m.data2 & 0x7F) == 0)
                        {
                            pedalOn = false;
                        }
                        else
                        {
                            pedalOn = true;
                        }
                    }
                }
            }
        }
    }
}
```

Appendix B

MATLAB code

B.1 Correcting the ground truth vector

Please note: this example aims for local maxima; the logic for local minima is the same.

```
while(k < (length(oldGT) - 1))
    % is the current sample a step occurrence?
    if (oldGT(k) == 1)
        % being a step occurrence, is the current sample also a local maximum?
        if (acc_yz_sq(k) > acc_yz_sq(k-1)) && (acc_yz_sq(k) > acc_yz_sq(k+1))
            % if it is, the ground truth is correct
            steps_vec(k) = 1;
        % but what happens if the current sample is not a local maximum?
        else
            % we search for local maxima at the left of this sample:
            min_L = 0;
            l = 1;
            while (min_L == 0)
                % if a local maximum is found at location k-l, save its index
                if ((acc_yz_sq(k-l) > acc_yz_sq(k-l-1)) && (acc_yz_sq(k-l) > acc_yz_sq(k-l+1)))
                    min_L = k - l;
                % if not, increase distance l
                else
                    l = l + 1;
                end
            end
            % and we also search for local maxima at the right of this sample:
            min_R = 0;
            r = 1;
            while (min_R == 0)
                % if a local maximum is found at location k+r, save its index
                if((acc_yz_sq(k+r) > acc_yz_sq(k+r-1)) && (acc_yz_sq(k+r) > acc_yz_sq(k+r+1)))
                    min_R = k + r;
                % if not, increase distance r
                else
                    r = r + 1;
                end
            end
            % then, the distances are compared and the nearest maximum is considered
            dist = min(l,r);
            if (dist == l)
                steps_vec(k-l) = 1;
            else
                steps_vec(k+r) = 1;
            end
        end
    end
    k = k + 1;
end
```

B.2 Refining the step detection threshold

Please note: this example aims for local maxima; the logic for local minima is the same.

```
DEF_WINDOW_HALF = 8; % defines a window of approximately 133ms
increment = 0.01;
threshold_yz = min(train_accyz_filtered);
max_threshold_yz = max(train_accyz_filtered);
tp = 0; tn = 0; fp = 0; fn = 0;
p = 0; r = 0; f1 = 0;
best_threshold = 0;
final_step_arr = zeros(length(train_accyz_filtered),1);

while (threshold_yz < max_threshold_yz)
    % first, get the predictions
    k = 2; % for looking one sample behind (peak detection)
    cnt = 0; % for counting ~1/3rd of a second after detection
    predicted_steps = zeros(length(train_accyz_filtered),1);
    while (k < length(train_accyz_filtered))
        if (cnt == 0)
            if ((train_accyz_filtered(k) > threshold_yz)... % val > threshold?
             && (train_accyz_filtered(k) > train_accyz_filtered(k-1))...
             && (train_accyz_filtered(k) > train_accyz_filtered(k+1))) % val is a peak?
                predicted_steps(k) = 1; % step detected!
                cnt = 1; % stop for ~1/3rd of a second
            end
        else
            if (cnt == 20)
                cnt = 0;
            else
                cnt = cnt + 1;
            end
        end
        k = k + 1;
    end

    k = DEF_WINDOW_HALF + 1; % so we can look X samples behind for GT positives
    k_end = length(predicted_steps) - DEF_WINDOW_HALF; % to look X samples further
    % start by filling tn, fn, fp and tp with matching values
    aux_tp = sum(predicted_steps==1&steps_vec==1);
    aux_tn = sum(predicted_steps==0&steps_vec==0);
    aux_fp = sum(predicted_steps==1&steps_vec==0);
    aux_fn = sum(predicted_steps==0&steps_vec==1);
    while(k <= k_end) % now check for correct predictions, just SLIGHTLY out of place
        if (predicted_steps(k) == 1 && steps_vec(k) == 0)
            if (sum(steps_vec(k-DEF_WINDOW_HALF:k+DEF_WINDOW_HALF)) ~= 0)
                aux_tp = aux_tp + 1;
                aux_tn = aux_tn + 1;
                aux_fp = aux_fp - 1;
                aux_fn = aux_fn - 1;
            end
        end
        k = k + 1;
    end
    aux_p  = aux_tp/(aux_tp+aux_fp);
    aux_r  = aux_tp/(aux_tp+aux_fn);
    aux_f1 = 2*(aux_p*aux_r)/(aux_p+aux_r);
    if (aux_f1 > f1)
        tp = aux_tp;
        tn = aux_tn;
        fp = aux_fp;
        fn = aux_fn;
        p = aux_p;
        r = aux_r;
        f1 = aux_f1;
        best_threshold = threshold_yz;
        final_step_arr = predicted_steps;
    end
```

```
    assert(aux_tp + aux_tn + aux_fp + aux_fn == length(predicted_steps));
    threshold_yz = threshold_yz + increment;
end

% now, let's see how the algorithm behaves on the test set
% first, get the predictions
k = 2; % for looking one sample behind (peak detection)
cnt = 0; % for counting ~1/3rd of a second after detection
test_predicted_steps = zeros(length(test_accyz_filtered),1);
while (k < length(test_accyz_filtered))
    if (cnt == 0)
        if ((test_accyz_filtered(k) > best_threshold)... % val > threshold?
         && (test_accyz_filtered(k) > test_accyz_filtered(k-1))...
         && (test_accyz_filtered(k) > test_accyz_filtered(k+1))) % val is a peak?
            test_predicted_steps(k) = 1; % step detected!
            cnt = 1; % stop for ~1/3rd of a second
        end
    else
        if (cnt == 20)
            cnt = 0;
        else
            cnt = cnt + 1;
        end
    end
    k = k + 1;
end
k = DEF_WINDOW_HALF+1; % so we can look X samples behind for GT positives
k_end = length(test_predicted_steps) - DEF_WINDOW_HALF; % to look X samples further
% start by filling tn, fn, fp and tp with matching values
test_tp = sum(test_predicted_steps==1&steps_test==1);
test_tn = sum(test_predicted_steps==0&steps_test==0);
test_fp = sum(test_predicted_steps==1&steps_test==0);
test_fn = sum(test_predicted_steps==0&steps_test==1);
while(k <= k_end) % now check for correct predictions, just SLIGHTLY out of place
    if (test_predicted_steps(k) == 1 && steps_test(k) == 0)
        if (sum(steps_test(k-DEF_WINDOW_HALF:k+DEF_WINDOW_HALF)) ~= 0)
            test_tp = test_tp + 1;
            test_tn = test_tn + 1;
            test_fp = test_fp - 1;
            test_fn = test_fn - 1;
        end
    end
    k = k + 1;
end
test_p  = test_tp/(test_tp+test_fp);
test_r  = test_tp/(test_tp+test_fn);
test_f1 = 2*(test_p*test_r)/(test_p+test_r);
```

This page intentionally left blank.

Appendix C

Java code

C.1 Java version of the *smflite* library

```
package org.keijiro.smflite;
import java.util.*;

public class MidiEvent {

    public byte status;
    public byte data1;
    public byte data2;

    public MidiEvent(byte status, byte data1, byte data2)
    {
        this.status = status;
        this.data1 = data1;
        this.data2 = data2;
    }

    public String ToString()
    {
        return "[" + String.format("%02x", status) + "," + String.format("%02x", data1) + "," +
            String.format("%02x", data2) + "]";
    }

}

public class MidiDataStreamReader {

    byte[] data;
    int offset;

    public int getOffset()
    {
        return offset;
    }

    public MidiDataStreamReader(byte[] data)
    {
        this.data = data;
    }

    public void Advance(int length)
    {
        offset += length;
    }

    public byte PeekByte()
    {
```

```
        return data[offset];
    }

    public byte ReadByte()
    {
        return data[offset++];
    }

    public char[] ReadChars(int length)
    {
        char[] temp = new char[length];
        for(int i = 0; i < length; i++)
        {
            temp[i] = (char)(ReadByte() & 0xff);
        }
        return temp;
    }

    public int ReadBEInt32()
    {
        int b1 = ReadByte() & 0xff;
        int b2 = ReadByte() & 0xff;
        int b3 = ReadByte() & 0xff;
        int b4 = ReadByte() & 0xff;
        return b4 + (b3 << 8) + (b2 << 16) + (b1 << 24);
    }

    public int ReadBEInt16()
    {
        int b1 = ReadByte() & 0xff;
        int b2 = ReadByte() & 0xff;
        return b2 + (b1 << 8);
    }

    public int ReadMultiByteValue()
    {
        int value = 0;
        while(true)
        {
            int b = ReadByte() & 0xff;
            value += b & 0x7f;
            if (b < 0x80)
            {
                break;
            }
            value <<= 7;
        }
        return value;
    }

}

public class DeltaEventPair {

    public int delta;
    public MidiEvent midiEvent;

    public DeltaEventPair(int delta, MidiEvent midiEvent)
    {
        this.delta = delta;
        this.midiEvent = midiEvent;
    }

    public String ToString()
    {
        return "(" + String.format("%02x", delta) + ":" + midiEvent + ")";
    }

}
```

```java
public class MidiTrack {

    List<DeltaEventPair> sequence;

    public MidiTrack()
    {
        sequence = new ArrayList<DeltaEventPair>();
    }

    public void AddEvent(int delta, MidiEvent midiEvent)
    {
        sequence.add(new DeltaEventPair(delta, midiEvent));
    }

    public ListIterator<DeltaEventPair> GetEnumerator()
    {
        return sequence.listIterator();
    }

    public DeltaEventPair GetAtIndex(int index)
    {
        return sequence.get(index);
    }

    public String ToString()
    {
        StringBuilder sb = new StringBuilder();
        for (DeltaEventPair pair : sequence)
        {
            sb.append(pair);
        }
        return sb.toString();
    }

}

public class MidiFileContainer {

    public int division;
    public List<MidiTrack> tracks;

    public MidiFileContainer(int division, List<MidiTrack> tracks)
    {
        this.division = division;
        this.tracks = tracks;
    }

    public String ToString()
    {
        String temp = division + ",";
        StringBuilder sb = new StringBuilder();
        sb.append(temp);
        for (MidiTrack track : tracks)
        {
            sb.append(track);
        }
        return sb.toString();
    }
}

public class MidiFileLoader {

    public static MidiFileContainer Load(byte[] data) {

        List<MidiTrack> tracks = new ArrayList<MidiTrack>();
        MidiDataStreamReader reader = new MidiDataStreamReader(data);

        // Chunk type
```

```
    if (!(new String(reader.ReadChars(4))).equals("MThd"))
    {
        throw new CustomException("Can't find header chunk!");
    }

    // Chunk length
    if (reader.ReadBEInt32() != 6)
    {
        throw new CustomException("Length of header chunk must be 6!");
    }

    // Format (unused)
    reader.Advance(2);

    // Number of tracks
    int trackCount = reader.ReadBEInt16();

    // Delta-time divisions
    int division = reader.ReadBEInt16();
    if ((division & 0x8000) != 0)
    {
        throw new CustomException("SMPTE timecode is not supported.");
    }

    // Read the tracks
    for (int trackIndex = 0; trackIndex < trackCount; trackIndex++)
    {
        tracks.add(ReadTrack(reader));
    }

    return new MidiFileContainer(division,tracks);

}

static MidiTrack ReadTrack (MidiDataStreamReader reader) {

    MidiTrack track = new MidiTrack();

    // Chunk type
    if (!(new String(reader.ReadChars(4))).equals("MTrk"))
    {
        throw new CustomException("Can't find track chunk!");
    }

    // Chunk length
    int chunkEnd = reader.ReadBEInt32();
    chunkEnd += reader.getOffset();

    // Read delta-time and event pairs
    int ev = 0;
    while (reader.getOffset() < chunkEnd)
    {
        // Delta time
        int delta = reader.ReadMultiByteValue();

        // Event type
        if ((reader.PeekByte() & 0x80) != 0)
        {
            ev = reader.ReadByte() & 0xff;
        }

        if (ev == 0xff) // 0xff: Meta event (unused)
        {
            reader.Advance(1);
            reader.Advance(reader.ReadMultiByteValue());
        }
        else if (ev == 0xf0) // 0xf0: SysEx (unused)
        {
            while ((reader.ReadByte() & 0xff) != 0xf7) {}
```

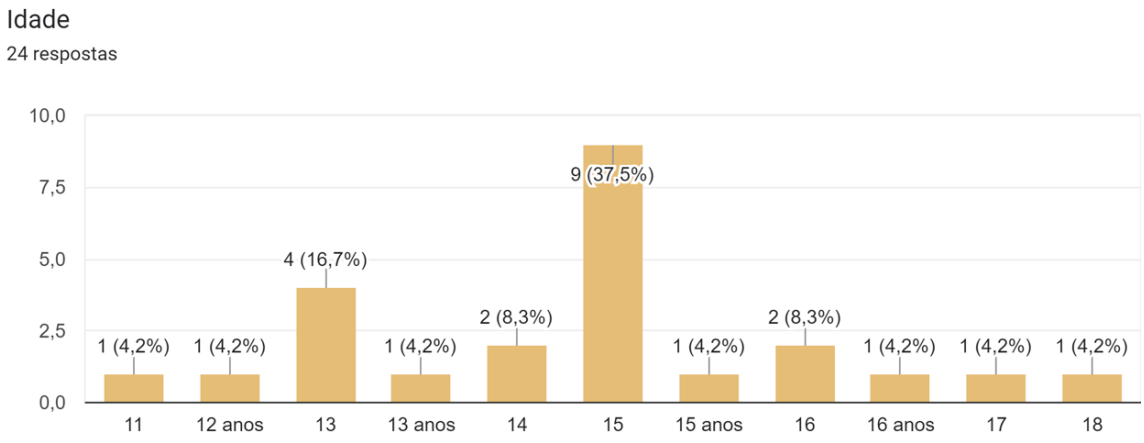
```
            }
            else // MIDI event
            {
                byte data1 = reader.ReadByte();
                byte data2 = ((ev & 0xe0) == 0xc0) ? (byte)0 : reader.ReadByte();
                track.AddEvent(delta, new MidiEvent((byte)ev, data1, data2));
            }

        }

        return track;
    }
}

public class MidiTrackSequencer {

    ListIterator<DeltaEventPair> enumerator;
    boolean playing;
    public float pulsePerSecond;
    float pulseToNext;
    float pulseCounter;
    DeltaEventPair pair;

    public boolean isPlaying()
    {
        return playing;
    }

    public MidiTrackSequencer(MidiTrack track, int ppqn, float bpm)
    {
        pulsePerSecond = bpm / 60.0f * ppqn;
        enumerator = track.GetEnumerator();
    }

    public List<MidiEvent> Start()
    {
        if (enumerator.hasNext())
        {
            pair = enumerator.next();
            pulseToNext = pair.delta;
            playing = true;
            return Advance(0);
        }
        else
        {
            playing = false;
            return null;
        }
    }

    public List<MidiEvent> Advance(float deltaTime)
    {

        if (!playing)
        {
            return null;
        }

        pulseCounter += pulsePerSecond * deltaTime;

        if (pulseCounter < pulseToNext)
        {
            return null;
        }

        List<MidiEvent> messages = new ArrayList<MidiEvent>();

        while (pulseCounter >= pulseToNext)
        {
```

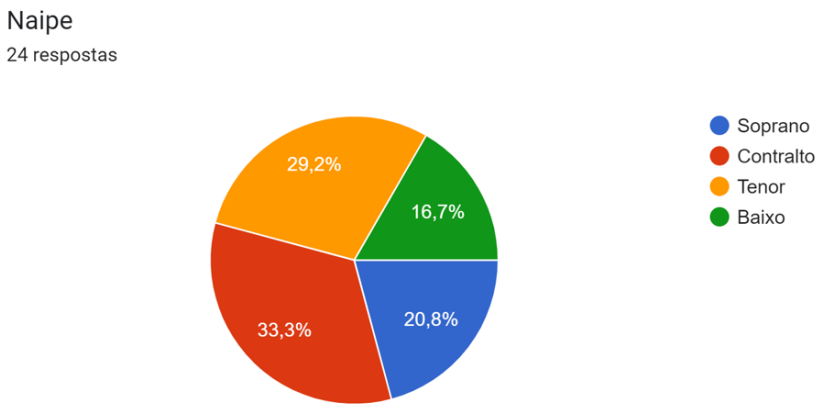
```
            messages.add(pair.midiEvent);

            if (!enumerator.hasNext())
            {
                playing = false;
                break;
            }
            pair = enumerator.next();
            pulseCounter -= pulseToNext;
            pulseToNext = pair.delta;
        }

        return messages;
    }

}

public class CustomException extends RuntimeException
{
    public CustomException(String errorMessage)
    {
        super(errorMessage);
    }
}
```

Appendix D

Questionnaire (in Portuguese)

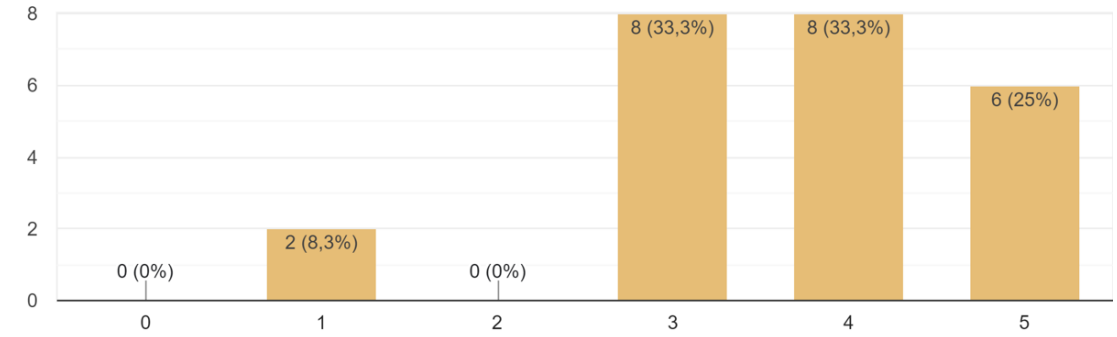
D.1 Multiple choice questions





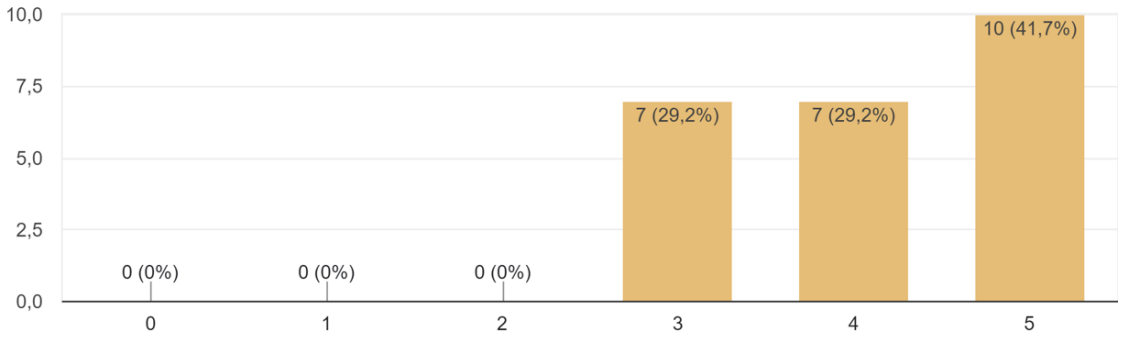
De 0 a 5, o quão atrativa é a interface de usuário?

24 respostas



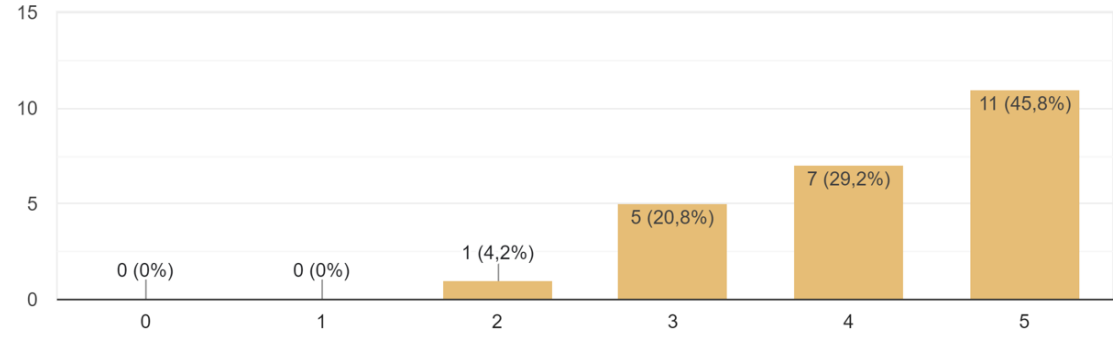
De 0 a 5, o quão fácil de utilizar são os controles da app?

24 respostas



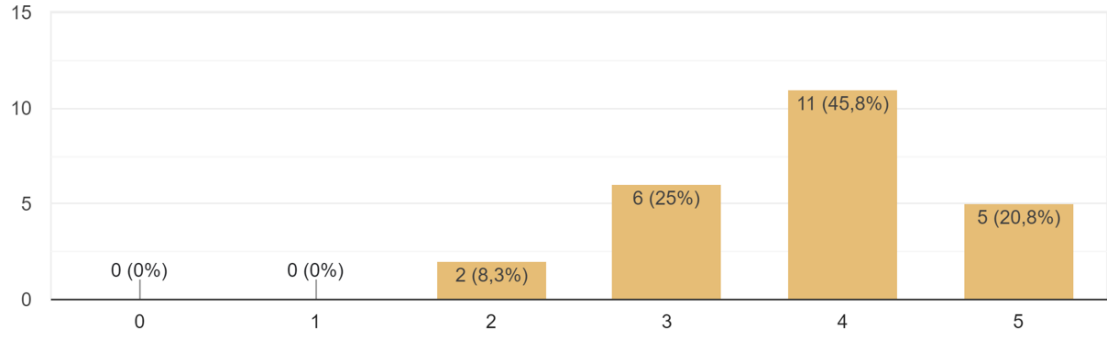
De 0 a 5, o quão interessante é esta forma de praticar música?

24 respostas



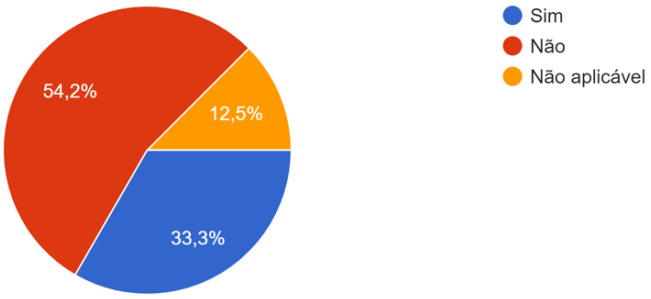
De 0 a 5, como avalia o desempenho de detecção de passo?

24 respostas



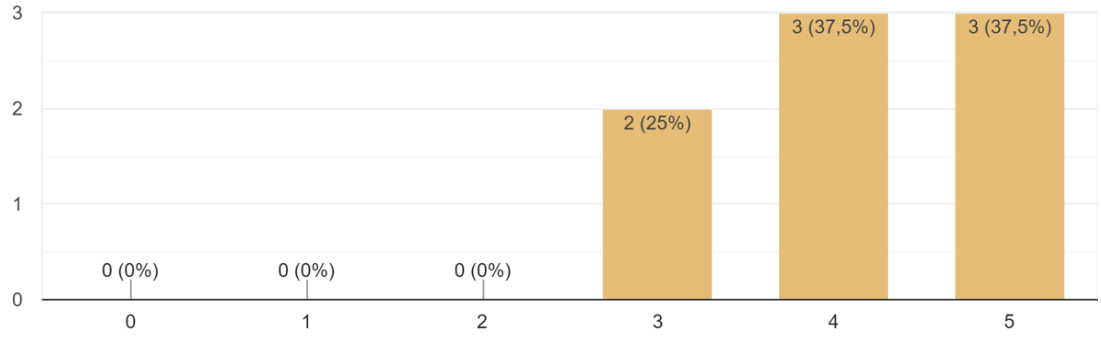
Foi necessário ajustar a sensibilidade de detecção de passo no slider azul?

24 respostas



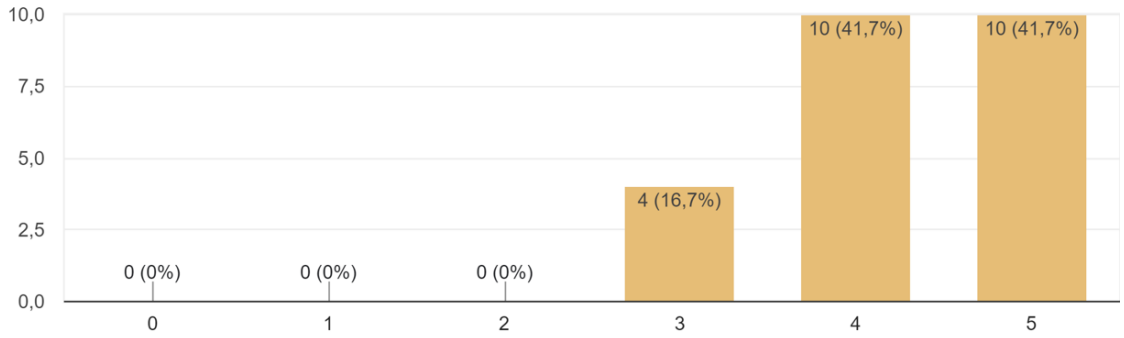
De 0 a 5, se ajustou a sensibilidade da detecção no slider azul, como avalia o desempenho do ajuste de sensibilidade?

8 respostas



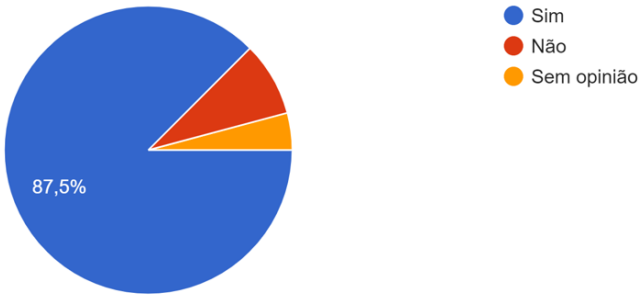
De 0 a 5, como classifica sua satisfação com a utilização do aplicativo, de um modo geral?

24 respostas



Considera esta aplicação uma inovação na prática musical?

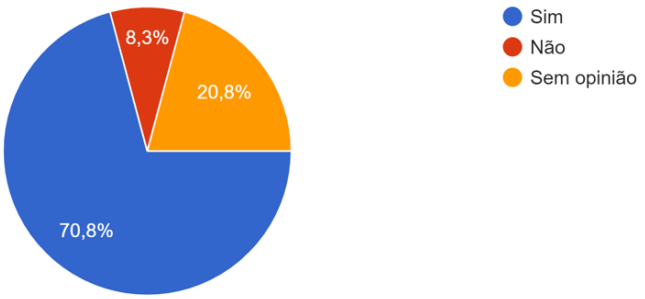
24 respostas



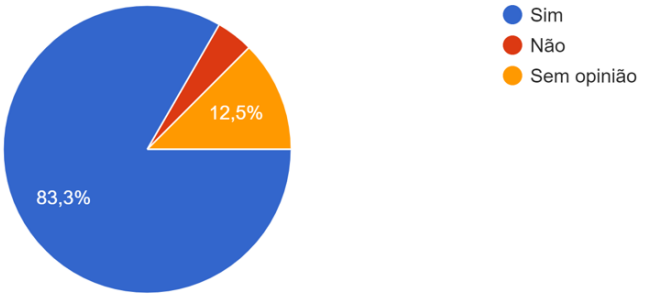
Considera este tipo de prática musical interessante?

24 respostas

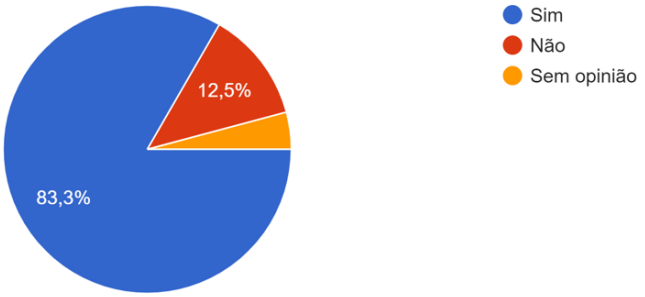
Sente que a utilização da app seria útil na aprendizagem de passagens musicais difíceis?
24 respostas



Sente que a utilização da app tornaria a sua experiência enquanto membro do coro mais interessante?
24 respostas

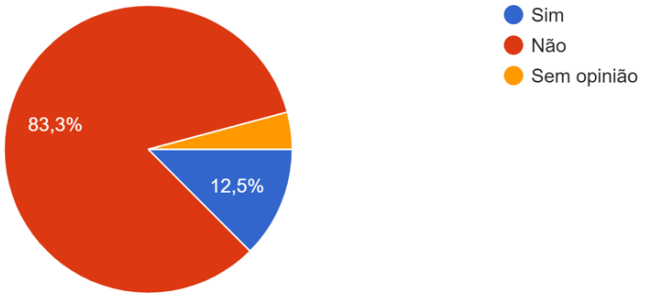


Caso esta aplicação fosse disponibilizada na Play Store (ou App Store), faria download da mesma e utilizá-la-ia, quer para fins de aprendizagem ou de lazer?
24 respostas



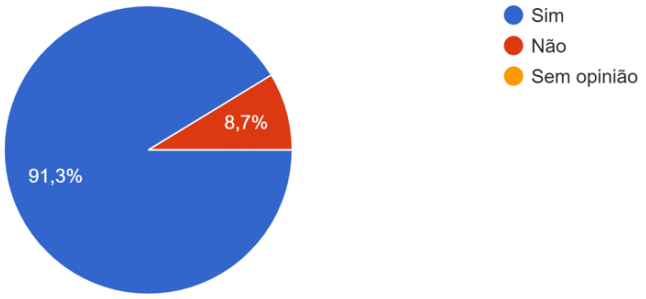
Considera que esta aplicação desvia o foco do utilizador na prática musical, acabando por ter um efeito negativo na aprendizagem?

24 respostas



Gostaria de ver esta aplicação mais desenvolvida e espalhada por um elevado número de utilizadores?

23 respostas



D.2 Suggestions and opinions

(1) *Achei uma plataforma muito legal e inovadora e com recursos diferentes !*

(2) *Muito legal o aplicativo, inovador!*

(3) *Funciona melhor se colocado no bolso*

(4) *Gostei muito do app, funcionou direitinho! Parabéns pelo trabalho! Minha sugestão é a que enviei para o Eric no WhatsApp, de talvez o app trabalhar também com a dinâmica da música. Com base no quão forte é o nosso passo fazer a marcação dos acentos agógicos, ou em movimentos como abaixar e levantar para fazer o "crescendo", por exemplo.*

(5) *Interessante, facilita na aprendizagem.*

(6) *Esse app e muito bom, ótimo de aprendizagem. Esse tipo de app pode ser utilizado por muitas pessoas interessadas na música*

(7) *Em geral o aplicativo é bastante eficiente no que propõe. É um ótimo meio para aprender, e ter noção corporal sobre ritmo (por exemplo, a pessoa caminharia no ritmo de cada música do repertório, tendo noção de andamento, tempo e até saberia diferenciar o ritmo de cada música). Algumas funções não me agradaram tanto assim, a sensibilidade não é tão boa, mesmo balançando forte meu celular, a sensibilidade não funcionou como deveria. Achei meio desmotivante o App só aceitar arquivos específicos de música e não todos.*

(8) *Sugestão: Coloque suporte ao mp3*

(9) *Gostei muito*

(10) *É um ótimo aplicativo e facilita muito pra quem não tem tempo para ensaiar as músicas em casa.*

This page intentionally left blank.

References

[1] Ç. Tan, M. Pamuk, and A. Dönder, «Loneliness and mobile phone», *Procedia - Social and Behavioral Sciences*, vol. 103, Nov. 2013. DOI: 10.1016/j.sbspro.2013.10.378.

[2] N. Guo, M. P. Wang, T. T. Luk, *et al.*, «The association of problematic smartphone use with family well-being mediated by family communication in Chinese adults: a population-based study», *Journal of Behavioral Addictions*, vol. 8, no. 3, pp. 412–419, 2019. DOI: 10.1556/2006.8.2019.39. [Online]. Available: https://akjournals.com/view/journals/2006/8/3/article-p412.xml.

[3] S. Qaisar, N. Akhter, A. Masood, and S. Rashid, «Problematic mobile phone use, academic procrastination and academic performance of college students», Dec. 2017.

[4] A. Shrivastava and M. Shrivastava, «Classroom distraction due to mobile phones usage by students: college teachers' perceptions», pp. 2279–0764, May 2014.

[5] M. Ansari and A. Tripathi, «An investigation of effectiveness of mobile learning apps in higher education in India», vol. 2, pp. 33–41, Mar. 2017.

[6] C. Liu, G.-J. Hwang, Y.-f. Tu, Y. Yin, and Y. Wang, «Research advancement and foci of mobile technology-supported music education: a systematic review and social network analysis on 2008-2019 academic publications», *Interactive Learning Environments*, pp. 1–20, Sep. 2021. DOI: 10.1080/10494820.2021.1974890.

[7] A. Romeo, E. G. Bryand, and P. R. Kane, «Electronic tablets and choral singing: initial investigations», Worcester Polytechnic Institute, Tech. Rep., Apr. 2015.

[8] L. M. Pedrero-Esteban, A. Barrios-Rubio, and V. Medina-Ávila, «Teenagers, smartphones and digital audio consumption in the age of Spotify», *Comunicar*, 2019.

[9] D. Kaufmann, «Reflection: Benefits of gamification in online higher education», vol. 7, Aug. 2018. DOI: 10.9743/JIR.2018.12.

[10] F. Styns, L. Van Noorden, D. Moelants, and M. Leman, «Walking on music», *Human movement science*, vol. 26, pp. 769–85, Nov. 2007. DOI: 10.1016/j.humov.2007.07.007.

[11] «XXIV Encontro da Associação Portuguesa de Engenharia de Áudio», *APEA-AES*, 2022. [Online]. Available: https://aes.org.pt/apea/encontro24/programa24.pdf.

[12] J. L. Aróstegui, «Exploring the global decline of music education», *Arts Education Policy Review*, vol. 117, pp. 96–103, Apr. 2016. DOI: 10.1080/10632913.2015.1007406.

[13] C. Ng and K. Hartwig, «Teachers' perceptions of declining participation in school music», *Research Studies in Music Education*, vol. 33, pp. 123–142, Dec. 2011. DOI: 10.1177/1321103X11423598.

[14] J. Sastre Martínez, J. Cerda, W. García, *et al.*, «New technologies for music education», Sep. 2013. DOI: 10.1109/ICeLeTE.2013.6644364.

[15] P. K. Ganesan and V. Raja, «Mobile learning», in Apr. 2019, pp. 97–105, ISBN: 978-93-88316-08-8.

[16] S. Criollo-C, A. Guerrero-Arias, A. Jaramillo-Alcázar, and S. Luján-Mora, «Mobile learning technologies for education: benefits and pending issues», *Open Journal of Applied Sciences*, vol. 11, Apr. 2021. DOI: 10.3390/app11094111.

[17] B. Arnold, «Gamification in education», *Annual American Society of Business and Behavioral Sciences Conference*, Jan. 2014.

[18] J. Lee and J. Hammer, «Gamification in education: what, how, why bother?», *Academic Exchange Quarterly*, vol. 15, pp. 1–5, Jan. 2011.

[19] I. Furdu, C. Tomozei, and U. Köse, «Pros and cons of gamification and gaming in classroom», *Broad Research in Artificial Intelligence and Neuroscience*, vol. 8, pp. 56–62, Aug. 2017.

[20] J. Rabah, R. Cassidy, and R. Beauchemin, *Gamification in education: real benefits or edutainment?*, May 2018. DOI: 10.13140/RG.2.2.28673.56162.

[21] S. Criollo-C, S. Luján-Mora, and A. Jaramillo-Alcázar, «Advantages and disadvantages of m-learning in current education», Mar. 2018, pp. 1–6. DOI: 10.1109/EDUNINE.2018.8450979.

[22] C. Gomes, M. Figueiredo, and J. Bidarra, «Gamification in teaching music: Case study», Mar. 2014.

[23] H. Birch and C. Brett, «Exploring a leaderboard alternative in a gamified mobile app for music learning», Apr. 2019. DOI: 10.33965/ml2019_201903L014.

[24] C. Berrol, «The neurophysiologic basis of the mind/body connection in dance/movement therapy», *American Journal of Dance Therapy*, vol. 14, pp. 19–29, Mar. 1992. DOI: 10.1007/BF00844132.

[25] A. Ravignani and G. Madison, «The paradox of isochrony in the evolution of human rhythm», *Frontiers in Psychology*, vol. 8, Nov. 2017. DOI: 10.3389/fpsyg.2017.01820.

[26] M. French, «Rhythmic movements and feeling states», Ph.D. dissertation, May 2018. DOI: 10.13140/RG.2.2.25553.86886.

[27] B. Moens, L. Van Noorden, and M. Leman, «D-jogger: syncing music with walking», *Proceedings of the 7th Sound and Music Computing Conference, SMC 2010*, Jan. 2010.

[28] V. Gikas and H. Perakis, «Rigorous performance evaluation of smartphone GNSS/IMU sensors for ITS applications», *Sensors*, vol. 16, no. 8, 2016, ISSN: 1424-8220. DOI: 10.3390/s16081240. [Online]. Available: https://www.mdpi.com/1424-8220/16/8/1240.

[29] Z. Saleh, «Artificial intelligence definition, ethics and standards», *Electronics and Communications: Law, Standards and Practice | 18ELEC07I*, Apr. 2019.

[30] J. Fürnkranz and T. Kliegr, «A brief overview of rule learning», Aug. 2015. DOI: 10.1007/978-3-319-21542-6_4.

[31] T. Tiwari, T. Tiwari, and S. Tiwari, «How artificial intelligence, machine learning and deep learning are radically different?», *International Journal of Advanced Research in Computer Science and Software Engineering*, vol. 8, p. 1, Mar. 2018. DOI: 10.23956/ijarcsse.v8i2.569.

[32] Y. LeCun, Y. Bengio, and G. Hinton, «Deep learning», *Nature*, vol. 521, pp. 436–44, May 2015. DOI: 10.1038/nature14539.

[33] N. Al Abiad, Y. Kone, V. Renaudin, and T. Robert, «Smartphone inertial sensors based step detection driven by human gait learning», in *2021 International Conference on Indoor Positioning and Indoor Navigation (IPIN)*, 2021, pp. 1–8. DOI: 10.1109/IPIN51156.2021.9662513.

[34] I. López-Nava, M. Garcia-Constantino, and J. Favela, «Recognition of gait activities using acceleration data from a smartphone and a wearable device», *Proceedings*, vol. 31, p. 60, Nov. 2019. DOI: 10.3390/proceedings2019031060.

[35] S. Y. Park, S. J. Heo, and C. G. Park, «Accelerometer-based smartphone step detection using machine learning technique», in *2017 International Electrical Engineering Congress (iEECON)*, 2017, pp. 1–4. DOI: 10.1109/IEECON.2017.8075875.

[36] G. M. G. Rodrigues, «Real-time step detection using unconstrained smartphone», Ph.D. dissertation, NOVA University Lisbon Repository (RUN), 2020. [Online]. Available: https://run.unl.pt/handle/10362/106544.

[37] «Unity user manual 2020.3 (LTS)», *Unity Technologies*, [Online]. Available: https://docs.unity3d.com/2020.3/Documentation/Manual/UnityManual.html.

[38] M. Rubo, «Apply 2nd order butterworth low-pass and high-pass filters or notch filters to data as they arrive», *GitHub*, [Online]. Available: https://github.com/mariusrubo/Unity-IIR-Realtime-Filtering.

[39] J. Rothstein, *MIDI: A comprehensive introduction*. USA: A-R Editions, Inc., 1992, ISBN: 0895792583.

[40] «Official MIDI specifications», *The MIDI Association*, [Online]. Available: https://www.midi.org/specifications.

[41] K. Takahashi, «A minimal class library for handling standard MIDI files on Unity.», *GitHub*, [Online]. Available: https://github.com/keijiro/smflite.

[42] «Meet Android Studio», *Android Developers*, [Online]. Available: https://developer.android.com/studio/intro.

[43] P. López-Cabanillas, «Sonivox EAS for Linux and Qt», *GitHub*, [Online]. Available: https://github.com/pedrolcl/Linux-SonivoxEas.

[44] B. Porr, «An IIR filter library written in Java.», *GitHub*, [Online]. Available: https://github.com/berndporr/iirj.