Harmonic interaction for monophonic instruments through musical phrase to scale recognition

Guy Sion

Master's programme in Music, Communication and Technology

Department of Music
Norwegian University of
Science and Technology

Department of Musicology
University
of Oslo

June 2020

Abstract

This thesis introduces a novel approach for the augmentation of acoustic instruments by providing musicians playing monophonic instruments the ability to produce and control the harmonic outcome of their performance. This approach is integrated into an interactive music system that tracks the notes played by the instrument, analyzes an improvised melodic phrase, and identifies the harmonic environment in which the phrase is played. This information is then used as the input of a sound generating module which generates harmonic textures in accordance with the identified scale. At the heart of the system is an algorithm designed to identify a scale from the played musical phrase. The computation relies on established music theory and is based on musical parameters retrieved from the performed melodic phrase. A database of audio recordings comprised of improvised phrases played by several saxophonists is used to test the algorithm. The results of the evaluation process indicate that the algorithm is reliable, and it can consistently recognize the scale of an improvised melody conveyed by a live musician. This discovery led to the exploration of the affordance to influence accompanying harmony by a monophonic line and integrating the phrase-to-scale match algorithm within an interactive system for music-making. By interacting and playing with the system using a repurposed controller mounted on the saxophone, performance strategies and practical ways are offered to play, modify, and further develop the system.

Acknowledgments

I would like to thank my supervisor, Stefano Fasciani, for his encouragement, guidance and direction through the development of the system and the writing processes.

To all the teachers and classmates at the MCT program, both in Trondheim and Oslo, for these inspiring two years. I have learned a great deal from all of you.

Special thanks to my tutor, colleague and friend Armando Gonzáles Sosto for showing me the way in the Max forest and being available for any question day and night.

My deep gratitude goes to my brother, Yoav Sion, for debugging my brain and code.

To all of the musicians involved, and that I have been fortunate enough to play with, listen to, learn and be inspired from, thank you for your help and your music.

To my parents that always believed in me and allowed me to follow my dream, thank you for being there every step of the way.

I am particularly grateful to Berit Reisel, for her mental and coffee support, and for her extreme generosity in sharing her home with me.

To the ones that sacrificed the most, Liza, Anna Matilda and Ben, thank you for your unconditional love and support, for letting me work and letting me finish.

Table of Contents

1. Introduction

This thesis presents a novel approach for controlling harmony, sometimes referred to as the "vertical" aspect of music, with its "horizontal" aspect, the melodic line. The suggested approach includes a software application, an algorithm, and a method for repurposing a video game controller, to track the pitch of a saxophone, capture and analyze an improvised musical phrase, determine the scale of the phrase, and use that output for music generation in an interactive manner.

Recent advances in the field of music production, and the technology available for anyone who wishes to produce music by electronic means, have enabled a wave of artists to develop a personal sound, produce their own music, invent instruments or write code to serve their artistic needs and aesthetics. Musicians are now able to invent and customize music applications to further their artistic research and remain original and inventive. Machine learning algorithms allow the musician of today to interact and play with artificial intelligence models with a great deal of communication and musical expression. The use of technology, in combination with traditional acoustic musical instruments in a wide range of musical genres, is becoming mainstream. Technological developments like the loop-pedal, the harmonizer effect, various instrument augmentation projects, and the invention of the electronic wind instrument controller, have helped the saxophone to maintain its place as one of the most popular instruments across musical genres. These advancements help it to remain at the forefront of bridging technology with acoustic musical instruments.

Instrumentalists in general, but more specifically, saxophonists, face several limitations when it comes to using technology when playing, whether it is performing a concert, recording in a studio, or practicing at home. The saxophone, for example, cannot be muted in a considerable way without affecting the timbre quality or the overall experience of playing the instrument, unlike the electric guitar or the trumpet.[1] A key challenge for saxophonists employing technology in a performance setting is the operation of additional devices, controllers, or interfaces while playing. They experience limited spare bandwidth since playing the saxophone requires using both hands, almost all fingers and the mouth. The solution for this is usually using foot-pedals or using the hands during musical pauses.

From a personal perspective, playing the saxophone for over three decades, and being involved with performing improvised music for the past 20 years, I found myself in search of ways that will allow for harmonic control. The saxophone is a monophonic wind instrument capable of producing only one note at a time (disregarding advance Multiphonics techniques), and since note sustain and tone quality are determined by the length of the air stream and the physical combination of the instrument and player, this can be seen as limiting at times. I found that playing or controlling harmony by using additional devices can be quite tricky when considering both the limited physical bandwidth and the attention required. My motivation when approaching this thesis was

[1] Yamaha Silent Brass - https://no.yamaha.com/no/products/musical_instruments/winds/silent_brass/index.html

to develop a tool for woodwind players that will enable monophonic instruments to control or play harmony and, at the same time, that would 'feel natural,' be intuitive and promote creativity.

In this paper, I will present an overview of past and recent state-of-the-art technologies and developments, designed explicitly for saxophonists who wish to extend their auditory outcome. After establishing the absence of a comprehensive music system meant for live performance, which allows for harmonic control by a melodic input, I will suggest and describe such a system design. Later, a primary portion of the system, an algorithm designed to match a musical scale to an improvised phrase, will be presented and evaluated. Also, a demo video is provided to show the complete system in action, including two sound generating modules.[2]

1.1 Research question and objectives

Motivated by the concept of controlling the harmonic output of an interactive music system with the melodic output of an improvised line, as well as identifying the lack of, and thus the need for a system that grants this type of interactivity, my research question becomes: Can a system and an algorithm be developed to successfully identify the scale of a musical phrase for collective music generation? By reflecting upon this question, my objectives were realized accordingly:

1. Developing and evaluating an algorithm that will successfully identify the scale of an improvised musical phrase played by a monophonic wind instrument.
2. Creating a database containing improvised phrases by several saxophone players in different keys and scales for evaluating the algorithm.
3. Integrating the scale recognition algorithm in a real-time interactive music system available to users and developers as free and open-source software.
4. Developing, exploring, and presenting a practical way to play and interact with the system.

1.2 Thesis contribution and system overview

The research and development presented in this thesis are directly related to several subjects that were taught and discussed during my studies at the Music, Communication, and Technology (MCT) master program at the University of Oslo (UiO) and the Norwegian University of Science and Technology (NTNU). Among those subjects are human-computer interaction, audio programming, and interactive music systems. With this thesis, my contribution to these fields primarily stands on the development of the Phrase to Scale Match (PSM) Algorithm. This algorithm analyzes a monophonic musical phrase of any length and outputs an estimated scale name that matches the input phrase from a dataset of 21 common scales. The algorithm calculates a matching scale based on several variables like the number of note repetitions, note duration, and other changeable weight-increasing factors for characteristic scale notes and note recurrences. Furthermore, a comprehensive system is presented for handling audio input, detecting pitch,

[2] Demo video of the system - https://youtu.be/u-ObVjojjyc

analyzing a musical phrase, and appropriating a retro game controller to be used as a control interface together with two sound generating modules.

In the proposed system, the data flows through the following submodules:

1. Analog to digital conversion of the microphone signal, which captures the musician's improvised phrase
2. Pitch tracking module
3. Buffer capturing the performed notes, including controls allowing the musician to start and terminate the capturing process
4. Scale recognition module, based on the method presented in this thesis
5. Musical applications using the output of the recognition module (drone and arpeggiator in the current version)

The proposed system and the scale recognition algorithm presented in this thesis offers a way to track the notes of a musical phrase, analyze it, calculate the tonality and scale, and output the result to several applications for music generation. The users can manipulate parameters of the system via a dedicated controller that is attached to the saxophone. The user interface (Figure 1) provides visual feedback of the audio input, detected notes, matched scale, bass note and pressed buttons of the controller.

Figure 1: The user interface of the system

2. Background

This chapter presents previous work and theoretical background related to the development of the system and algorithm presented in this thesis. Designing a music system that interacts with a musician via melody and harmony and developing an algorithm to identify the scale of a single melodic line without modulation requires reviewing research from a wide range of fields. Since the system is grounded in work from various fields, the chapter is organized as follows. The first section presents a brief description of electric wind controllers and saxophone augmentation projects. Followed by a section discussing pitch tracking in general and, more specifically, the yin algorithm and the yin object. The following section provides theoretical music concepts used in scale recognition, together with a review of techniques and applications that deal with tonal frameworks. I will later describe some MIR methods and algorithms related to the Phrase-Scale Match algorithm (PSM). The last section of the chapter is a summary where I will present my motivation as it is based on prior work in the field. I will finish by presenting the research question together with my contribution.

2.1 Electric Wind Controller and Saxophone augmentation

In the 1930s, American radio engineer and inventor Benjamin F. Miessner developed an electroacoustic clarinet that featured an electromagnetic pickup for the reed vibration. Registering his patent in 1936 marked the start of nearly a decade of innovation of enhancing wind instruments by electronic means. Since then, there have been numerous endeavors with various degrees of success, to augment acoustic wind instruments electronically or built electric wind instruments based on acoustic ones. Almost 40 years later, the Lyricon[3] wind controller came to the market and was well-received by woodwind players. The Lyricon features a fingering system and mouthpiece setup based on the saxophone. It is highly expressive thanks to the ability to interpret reed articulation, breath-controlled dynamics, and embouchure-controlled pitch variation. The Lyricon has set the standard for hardware-based wind controllers, serving as the foundation for today's modern MIDI wind controllers. Since the early 1980s, digital wind controllers have gained more traction and popularity. Controllers such as the WX series[4] by Yamaha and EWI[5] by Akai are capable of generating a standard MIDI data stream, allowing the control of any MIDI-compatible synth with a high level of expressivity.

In the early 1980s, the Synthophone was the first attempt to augment an actual saxophone body, converting it into a midi controller by using various sensors. As stated by Burtner (2002), the developers of the Synthophone wanted to preserve the tactile interface of the saxophone and were willing to sacrifice its actual acoustic sound. The Synthophone[6] is considered to be the first MIDI saxophone controller. It does not produce any sound of its own, and the saxophone body function

[3] Lyricon Wikipedia page - https://en.wikipedia.org/wiki/Lyricon
[4] Yamaha WX5 - https://usa.yamaha.com/products/music_production/midi_controllers/wx5/index.html
[5] AKAI EWI series - https://www.akaipro.com/products/ewi-series
[6] Synthophone Zone - http://synthophone.info/indexh.htm

as a housing unit for the electronics. The MIDI data can be sent to synthesizers and various sound modules (Softwind Instruments, 1986; Andreas, 1988)

The EMEO is being marketed as a digital practice horn for saxophone players. Just like the Synthophone, the EMEO[7] is built up from an actual acoustic saxophone, but without the bell of the horn. The saxophone itself does not produce any sound but can connect via Bluetooth or USB cable to any DAW on smartphones, tablets, and computers to be used with VSTs for sound production.

Since the 1990s, there have been several small-scale attempts of saxophone augmentation, using a variety of techniques meant to serve different artistic and esthetic goals. I will now present a short review of the main saxophone augmentation projects:

Since 1997, Matthew Burtner (2002) has been developing the Metasaxophone[8], a "tenor saxophone fitted with an onboard computer microprocessor and an array of sensors that convert performance data into MIDI control messages." His primary motivation was to "put signal processing under direct expressive control of the performer." While maintaining the full acoustic functionality of the saxophone, the developer's dedicated software can interpret the sensory data, and through MIDI protocol, can directly control digital signal processing. The sensors, located on the saxophone, are continuous voltage force sensing resistors (FSR), five triggers, and an accelerometer. The sensory data is mapped to control deferent parameters like reverb, delay, noise generators, and filter parameters of sound modules in Max/MSP.

The Gluisax was developed by the Australian experimental electronic 'Bent Leather Band'. It is a collection of three augmented and meta saxophone interface/instrument. Inspired by Matthew Burtner's Metasaxophone, and by Schiesser and Traube's saxophone project (Schiesser and Traube, 2006), the Gluisax developers were interested in creating playable instruments that are expressive, responsive, versatile and practicable. They mounted on the saxophone a joystick, dial knobs, force-sensitive resistors (FSR) and microphones connected to Sukandar Kartadinata's Gluion interface to create 'OSC saxophones'. The OSC data is streamed into Max/MSP for controlling pitch transposition, delay time, comb filters, rhythmic looping and re-sampling (Favila, 2008).

Developed at Dongguk University in Seoul, Korea, the Telesaxophone is created for saxophone players and multimedia performing artists. It is a hybrid saxophone interface consisting of an original saxophone neck and mouthpiece. The developers' goal was to create an interface that would be comfortable to control and play, just like a real acoustic saxophone. An additional goal was to be able to control various multimedia works by playing the Telesaxophone. It consists of 14 button sensors, three dial sensors and sound sensors to detect the sound of the original mouthpiece and neck in real-time. An integrated Arduino board is used to obtain sensor data,

[7] The EMEO site - https://emeo.biz/
[8] Metasaxophone Systems - https://ccrma.stanford.edu/~mburtner/metasax.html

fingering and controlling recognition. The data is transmitted to the Max/MSP programming environment for musical sound synthesis and processing, as well as controlling mapped parameters of media artworks (Hong and Kim, 2017).

The Gest-O was developed in Columbia and presented at NIME 2012. It is an open-source tool for controlling the sound of the saxophone via the gestures of the performer. It is a hardware system containing an accelerometer and gyroscope sensors, connected via Bluetooth to a digital sound processing (DSP) system developed in Pure Data. Gestures are mapped to various effect parameters like Grane Sampling/Amplitude/Size, Ring Modulation, Reverb, Delay and Multiphonics. The developers investigated strategies to interpret and map gestures of a specific performer playing a specific piece. (Melo, Gómez and Vargas, 2012)

Dr. Saxophone is a Hybrid Saxophone Interface developed in Korea and presented at the ICSAI in 2016. The interface consists of a mainboard placed inside the saxophone's bell. It holds a tilt sensor and a pressure sensor (referred to as a sound sensor). Two switches and a dial have been installed instead of the last key of the saxophone (the last tone-hole, Bb). Having the mainboard inside the bell and modifying the Bb key compromises the acoustic sound of the saxophone. Data collected from the tilt sensor, pressure (sound) sensor, dial, and switches, is sent from an Arduino board via wireless technology (X-Bee pro), to sound effect parameters (reverb, delay, chorus) in Max/MSP (Hong, Kim and Han, 2016).

HypeSax[9] is a saxophone augmentation project in which a hybrid system enables an acoustic integration between the acoustic sound of the saxophone and electronics. Developed in New Zealand and presented at NIME 2019, the HypeSax system consists of several modular components attached to an alto saxophone. The system can work with some or all of its components. In addition to the touch, gyroscope and accelerometer sensors, pushbuttons, and a microphone, the developers introduce a new design of a saxophone's mouthpiece that holds a barometric sensor measuring air pressure. Another interesting feature of the HypeSax is the Un-mute unit, located inside the bell of the saxophone. The Un-mute is a self-contained audio system (soundcard and speaker) that allows the performer to add additional sound components into the final sound. The use of a speaker and the acoustic sound of the saxophone helps to achieve sound hybridization. It solves the common problem of sound source disembodiment, evident in many augmentation projects. The HypeSax connects via USB to a computer or a MIDI device. A server application was developed in Max/MSP to handle the sensory data and re-route it using either MIDI, OSC, Serial, or Max's send/receive messages. The HypeSax is currently under development, and research regarding the use of audio components like feedback, additive synthesizer, and effects is still ongoing (Flores, Murphy and Norris, 2019).

[9] New Music Technology | HypeSax - https://www.hypesax.com/

2.2 Pitch-tracking

When designing an interactive music system that can analyze the melodic output of the saxophone in a live performance situation, the pitch-tracking component of the system must be reliable, robust and precise. The system presented in this thesis is heavily based on correctly detecting the notes of a live-played musical phrase rooted in a set key and scale. Avoiding initial pitch-tracking errors and being suitable for an interactive music application, the pitch-tracking algorithm should meet these four conditions: First, the algorithm must have the ability to function in real-time. Second, it should have as low as possible output delay (latency). Third, it should be accurate in a noisy environment. Forth, the algorithm should be sensitive to the musical requirements of the performance (De La Cuadra, Master and Sapp, 2001).

2.2.1 Pitch detection algorithms

Pitch Detection Algorithms (PDA) are designed to estimate the fundamental frequency (f_0) of a quasiperiodic signal. PDAs are being applied in a range of fields dealing with speech (e.g., phonetics, speech coding, voice recognition, speech analysis-synthesis (vocoder) systems) and music (e.g., music information retrieval, musical performance systems, auto-tuning, beat detection, automatic score transcription). Several standard methods based on various mathematical principles are used to extract f_0. Pitch is a perceptual quantity related to the fundamental frequency of a periodic or pseudo-periodic waveform; therefore, it should be sufficient to determine the period of such oscillation, the inverse of which is the frequency of oscillation. In a noisy environment or when more than one instrument is playing, and the waveform consists of more than a simple sinusoid, the appearance of pitch becomes less clear. This makes it more difficult to estimate the pitch correctly (Gerhard, 2003).

The methods in which pitch detection algorithms operate can be divided into three groups (Cook, 1992). The first group of pitch detection methods operates in the time-domain. “The theory behind these methods is that if a waveform is periodic, then there are extractable time-repeating events that can be counted, and the number of these events that happen in a second is inversely related to the frequency” (Gerhard, 2003). This group includes methods that use the detection and timing of some time-domain features and methods that use autocorrelations functions or different norms to detect similarities between the waveform and a time-lagged version of itself. In this method, the signal is usually preprocessed to accentuate some time-domain feature, then the time between occurrences of that feature is calculated as the period of the signal. A typical time-domain feature detector is implemented by low pass filtering the signal, then detecting peaks or zero crossings. The second group of pitch detection methods operates in the frequency-domain, where the signal is converted from its original domain of time and space to a representation in the frequency domain. The frequency-domain representation is inspected for the first harmonic, the greatest common divisor of all harmonics, or other such indicators of the period. To avoid spectral smearing, sometimes defined as spectral leakage, the process of windowing the signal is recommended. Windowing consists of multiplying the time-domain signal by a finite-length

window with an amplitude that varies smoothly and gradually toward zero at the edges. This smooths the endpoints of the waveform, resulting in a continuous waveform without sharp transitions. This technique is also referred to as applying a window (Lyons, 2004). To easily find frequency domain features, various linear preprocessing steps can be used, for example, performing a linear prediction on the signal and using the residual signal for pitch tracking. In addition, non-linear operations like peak limiting can also simplify the location of harmonics. The third group of pitch detection methods uses a combination of time and frequency-domain techniques to detect pitch (Cook, 1992). All three groups of pitch tracking methods follow these three steps: preprocessing (filtering, frames splitting), searching for a possible value for $f_{0,}$ and tracking - following the choice of the most probable f_0 trajectory. Gerhard (2003) presents and survey standard pitch detection techniques and current state of the art pitch detection technology, and categorize them as such (Figure 2):

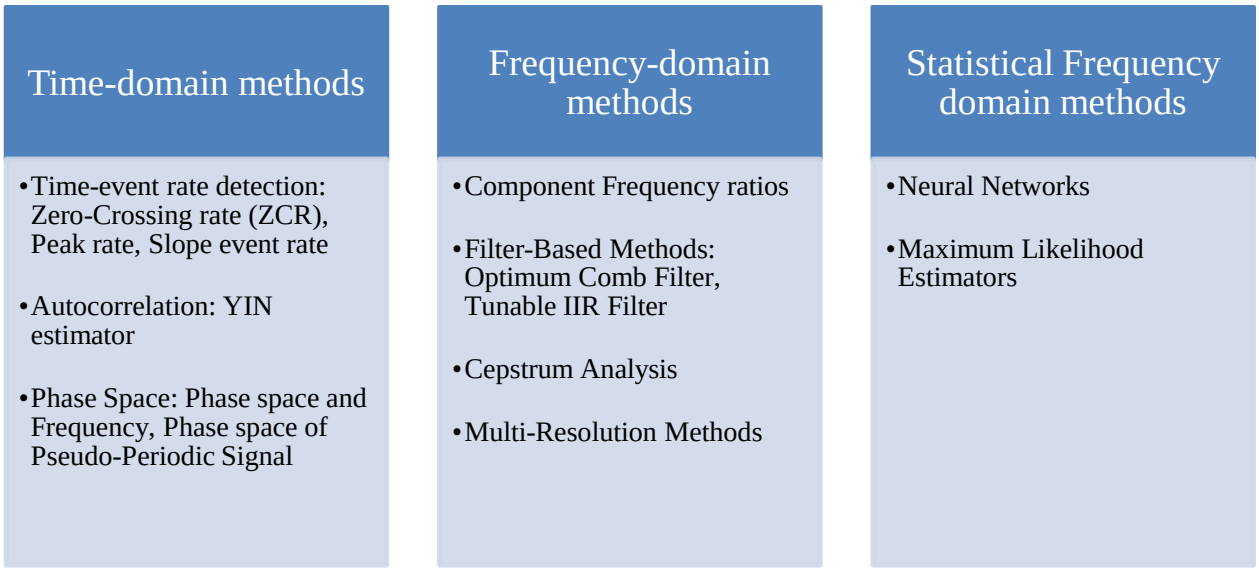


Figure 2: Pitch-detection techniques categorization by Gerhard (2003)

The pitch detection algorithm employed in my system is the YIN fundamental frequency estimator developed by Alain de Cheveigné and Hideki Kawahara. It is an off-the-shelf f_0 estimator that is available as a Max/MSP object from the 'Max Sound Box'[10] for real-time interaction with Max modules. The YIN estimator is based on the well-known autocorrelation method with several additional modifications that combine to prevent errors. In the sections below, I will discuss the autocorrelation method, the YIN f_0 estimator and the reasoning for the implementation of this specific method within my system.

2.2.2 Autocorrelation method

Autocorrelation is a method related to features detection in the time-domain. Measuring the correlation between two waveforms is a way to measure their similarity. "The waveforms are

[10] Max Sound Box | Ircam Forum - https://forum.ircam.fr/projects/detail/max-sound-box/

compared at different time intervals, and their "sameness" is calculated at each interval. The result of a correlation is a measure of similarity as a function of the time lag between the beginnings of the two waveforms. The autocorrelation function is the correlation of a waveform with itself" (Gerhard, 2003). When measuring the correlation of a waveform with itself, we expect exact similarity at a time lag of zero and an increased dissimilarity as the time lag increases.

Mathematically, the autocorrelation corresponding to a delay time *x* is calculated by:

1. finding the value of the signal at a time n
2. finding the value of the signal at a time n + v
3. multiplying those two values together
4. repeating the process for all possible times, n, and then
5. computing the average of all those products

The process can be repeated for (all) other values of *v*, resulting in an autocorrelation, which is a function of the delay time *v*. The mathematical definition of the autocorrelation function for an infinite discrete function $x[n]$ is shown in Equation 1

$$R_x(v) = \sum_{n=-\infty}^{\infty} x[n]x[n+v] \tag{1}$$

The mathematical definition of the autocorrelation function of a finite discrete function $x'[n]$ of size *N* is shown in Equation 2

$$R_{x'}(v) = \sum_{n=0}^{N-1-v} x'[n]x'[n+v] \tag{2}$$

The cross-correlation between two functions $x[n]$ and $y[n]$ is calculated using Equation 3

$$R_{xy}(v) = \sum_{n=-\infty}^{\infty} x[n]y[n+v] \tag{3}$$

Applying autocorrelation on periodic waveforms results in a more accurate estimation of the pitch. However, problems with this method arise when the autocorrelation of a harmonically complex pseudo-periodic waveform is being measured. Another difficulty with autocorrelation techniques is that peaks occur at sub-harmonics as well, making it difficult to distinguish between fundamental frequency and harmonics or partials. The YIN estimator addresses these problems (Gerhard, 2003).

2.2.3 YIN – a fundamental frequency estimator

The YIN f_0 estimator is an algorithm developed by De Cheveigné and Kawahara (2002). The algorithm is named after the Taoist "yin-yang" philosophical principle of balance, referring to the interplay between autocorrelation and cancellation it involves. In order to address the obstacles of using autocorrelation and reduce error rate, "YIN is based on the *difference function* (Equation 4) which, while similar to the autocorrelation, attempts to minimize the difference between the waveform and its delayed duplicate instead of maximizing the product (for autocorrelation)" (Gerhard, 2003).

$$d_t(\tau) = \sum_{j=1}^{W} (x_j - x_{j+\tau})^2 \tag{4}$$

In order to reduce the occurrence of subharmonic errors, YIN employs a cumulative mean function (Equation 5) which de-emphasizes higher-period dips in the difference function:

$$d'_t(\tau) = \begin{cases} 1, & \tau = 0 \\ \dfrac{dt(\tau)}{\frac{1}{\tau}\sum_{j=1}^{\tau} dt(j)} & otherwise \end{cases} \tag{5}$$

The authors describe YIN's method for f_0 estimation in six steps that build upon one another. Here is an abridged version of these steps:

Step 1: The autocorrelation method - finding the correlation of the signal with its delayed duplicate by a lag within a window with the autocorrelation function (ACF). Concluding that the ACF is quite sensitive to amplitude changes, which encourages the algorithm to choose a higher-order pick and make a too low of an error. The authors note that the "autocorrelation method makes too many errors" and offer further steps to reduce the error rate.

Step 2: Difference function - modeling the signal in the form of a difference function by using amplitude as a bias translates into a significant decrease in error rate (from 10.0% for the unbiased autocorrelation function to 1.95% with the difference function).

Step 3: Cumulative mean normalized difference function – replacing the difference function with a cumulative normalized difference, which de-emphasizes higher-period dips by avoid selecting values with zero lag. Lowering the sensitivity of the signal to amplitude modulations by introducing normalization makes the peaks more apparent than with the traditional autocorrelation method.

Step 4: Absolute threshold - setting an absolute threshold to avoid the subharmonic error (sometimes referred to as the octave error). This step allows us to pick a number (threshold) that suits the approximate expected noise level.

Step 5: Parabolic interpolation – is independent from other steps, although it relies on the spectral properties of the ACF (step 1). Parabolic interpolation is applied to approximate the minimum of the sampling period. Each local minimum of $d'(\tau)$ and its immediate neighbor is fit by a parabola, and the ordinate of the interpolated minimum is used in the dip-selection process. This results in a reduced fine error at all f_0 and avoided gross error at high f_0.

Step 6: Best local estimate - is reminiscent of median smoothing or dynamic programming techniques but differs in that it considers a relatively short interval and bases its choice on quality rather than mere continuity. Applying the best local estimate function helps to avoid rapid fluctuation on the time scale of the fundamental period, ensuring stable f_0 estimation.

De Cheveigné and Kawahara (2002) summarize their method as such:

> The combination of steps 1-6 constitutes a new method (YIN)… It is worth noting how the steps build upon one another. Replacing the ACF (step 1) by the difference function (step 2) paves the way for the cumulative mean normalization operation (step 3), upon which are based the threshold scheme (step 4) and the measure $d'(\tau)$ that selects the best local estimate (step 6). Parabolic interpolation (step 5) is independent from other steps, although it relies on the spectral properties of the ACF (step 1).

For a complete discussion of this method, including computational implementation and evaluation, please refer to the cited paper and the YIN algorithm documentation.[11]

2.2.4 YIN evaluation in previous work

After reviewing prior research evaluating the YIN method, it is evident that a large portion of the work has been done about voiced speech (De Cheveigné and Kawahara, 2002; Suk, Chung and Kojima, 2007; Zahorian and Hu, 2008; Ghahremani *et al.*, 2014). Formal evaluation of the YIN method with music has been limited and mostly informal (Gerhard, 2003; von dem Knesebeck and Zölzer, 2010; Babacan *et al.*, 2013; Robertson, 2014; Gao, 2015; Vasilik, Stillings and Cortazar, 2015). However, by examining previous research, one can quickly determine that the Yin method is regarded as reliable and accurate for both speech and music, and the Yin algorithm is considered a top-tier algorithm for f_0 estimation among peers. Gao (2015), as shown that the YIN algorithm can successfully retrieve all the notes in a musical phrase with note accuracy reaching 96.88%. Babacan *et al.* (2013) concluded that the YIN algorithm "achieved the best accuracy" among different pitch tracking techniques when compared against a database of singing sounds. De Cheveigné and Kawahara (2002) recognize that the "difficulties specific to music are the wide range and fast changes in f_0," to which I would point out the challenge of a noisy signal in a real-time performance setting. Although they state that "YIN has been only informally evaluated on music" and that other potential advantages of YIN like low latency "are yet to be tested", they "expect that it is appropriate for the task (of detecting pitch in music). "

[11] YIN algorithm documentation - http://mroy.chez-alice.fr/yin/index.html

2.2.5 Yin~ in Max

Based on the work by De Cheveigné and Kawahara (2002), a Max abstraction object has been developed by Norbert Schnell at the Institute for Research and Coordination in Acoustics/Music (IRCAM[12]) as part of the Ircam Real-Time Musical Interaction (IMTR[13]) research and development team. Available for free use from the Max Sound Box Library[14], the yin~ object is part of a Max collection of externals for real-time interaction, real-time analysis, synthesis and transformation of sound. By examining the help information of the yin~ object (Figure 3), one can identify the YIN algorithm parameters that can be fine-tuned.

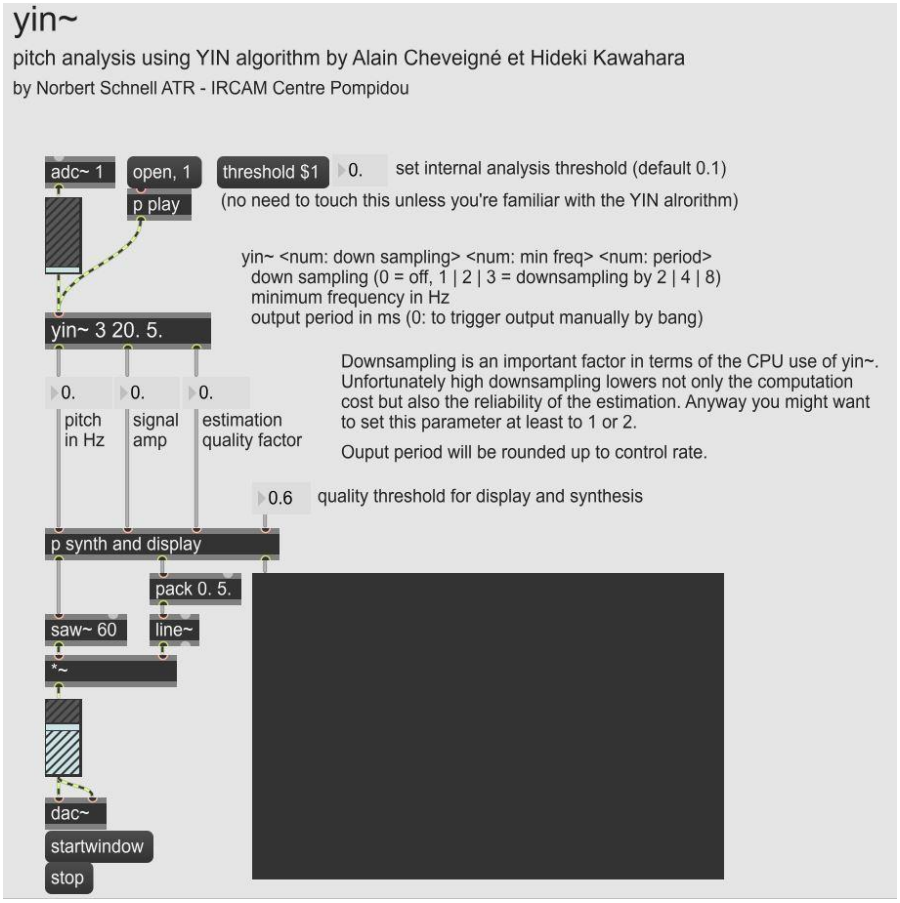


Figure 3: yin~ max help

The yin~ object receives a signal input. It has four parameters (represented by numbers) that can be adjusted to balance between successful pitch prediction and computational power: downsampling, minimum frequency, output period, and threshold. Downsampling can be done by 2(1), 4(2), 8(3) or none (0), and it is essential to keep in mind the tradeoff that "high downsampling lowers not only the computation cost but also the reliability of the estimation" as stated in the help

[12] Home | Ircam - https://www.ircam.fr/
[13] IRCAM Real-Time Musical Interactions - http://imtr.ircam.fr/imtr/IRCAM_Real-Time_Musical_Interactions
[14] Max Sound Box | Ircam Forum - https://forum.ircam.fr/projects/detail/max-sound-box/

file by the developer. The Yin algorithm does not have an upper limit on the frequency search range, but a lower limit can be set (in Hz) to better pitch estimation and conserve computation power. Setting how often the yin~ object will be updating its output values can be adjusted with the output period (in milliseconds). The threshold parameter (set to 0.1 as default) relates to the absolute threshold (step 4) of the Yin algorithm, helping us to set the approximate noise level of the signal.

There are three output values from the yin~ object: pitch, amplitude, and estimation quality factor. The pitch estimation is represented in Hz and will later be converted into a discrete symbolic form (MIDI). The signal amplitude, presented as a number between 0-1, will later assist in determining note repetition and other factors of the musical phrase (see Chapter 3). The estimation quality factor tells us how confident the yin~ is with its estimation of the pitch (0-not sure, 1-very sure). Setting a quality threshold will help us discriminate between noise and music.

De La Cuadra et al. (2001) have stated that "No pitch algorithm can possibly cover all requirements and unanticipated conditions in interactive music performance," however, by understanding how the YIN estimator operates, taking into consideration the musical and acoustical conditions it is applied in, and tuning its parameters accordingly, the result can be surprisingly sufficient.

2.3 Scale recognition

This sub-chapter provides background related to the technology to implement musical concepts with a music program. Identifying the key and scale of a monophonic musical phrase, composed or improvised, requires accommodating the practice of fundamental musicianship to a computer or a program. In the following, I present sections about music theory, tonality and several other musical concepts related to the Phrase-Scale Match (PSM) algorithm. Several existing methods for scale recognition are discussed as well. Formalizing musical concepts for a machine is a subject that involves research in the fields of music theory, music cognition, and artificial intelligence. The reviewed work and music theory sections described here are delimited by the framework of the offered algorithm and in no way forms a comprehensive overview of this broad field of machine musicianship.

When discussing algorithmic analysis in his book 'Machine musicianship', Rowe (2004) manages to illustrate some of the main processes that can be applied to real-time analysis of musical input (Figure 4). Rowe talks about different levels of algorithmic processes, whereas pitch input and pitch tracking are first forwarded to lower-level processes like finding the root, key, or identifying chords. Higher-level processes like segmentation, style recognition, or pattern processing are based on the output of those lower-level analyses and come later in the process. The phrase to scale algorithm attends to low-level analyses of key and scale identification of a composed or improvised musical phrase played by a single melodic line. The PSM algorithm receives an array of any number of notes (MIDI integers), each with its specific duration and in the order they were just played. Within a few milliseconds, the algorithm is expected to complete the analysis of the

played phrase and to output its key and scale. Later, that information will be used for generating music together with an agent.

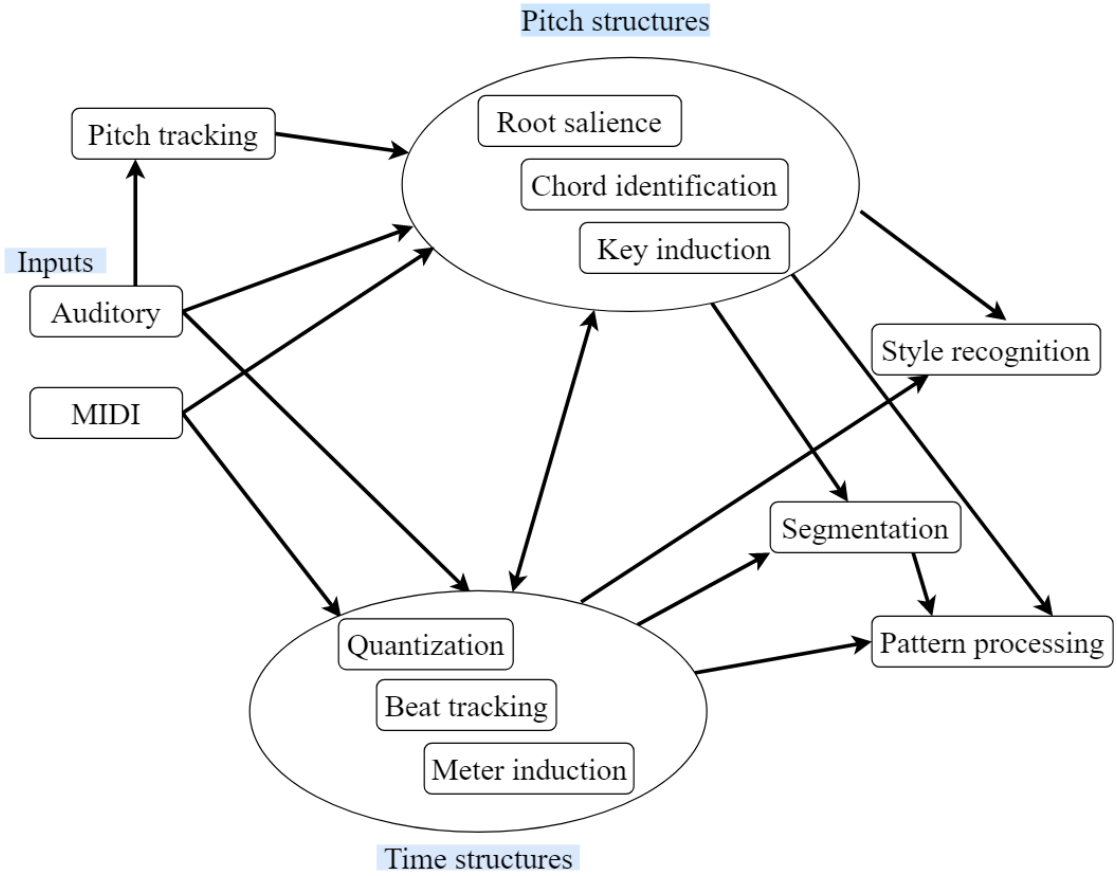


Figure 4: main processes for real-time analysis

2.3.1 General and related music theory concepts

Sound is made of vibrations and our ears and brain interpret those vibrations. When the vibrations are faster, we hear them as being higher and when the vibrations are slower, we hear them as being lower. If the vibrations happen in a consistent rate, we perceive them as having a consistent pitch or frequency or a note. The vast majority of music from the 1600s onwards (i.e., European music, contemporary classical music, popular music, jazz) is made by using twelve notes, and by considering how those notes relate to each other. The meaning of tonality can be broad and has been identified and explained in many ways. Some refer to tonality as any systematic organization of pitch or the relation between the tones of a scale or a musical system; others, as any rational and self-contained arrangement of musical pitch. Too often, and incorrectly, the word tonality functions as a synonym for "key". In the context of the PSM algorithm, finding the tonality of a melodic phrase means identifying the mode, a group of pitches, or scale, that forms the basis of the phrase. Diatonic or heptatonic scales in traditional western music consist of seven notes within one octave (Latin: *octavus*: eight). From each of the seven notes, a mode can be constructed. A mode is a system of notes, a scale, coupled with a set of characteristic melodic behavior. In use

since the middle ages and inspired by the theory of ancient Greek music, each mode consists of a unique interval sequence and has its own name. Modes that are constructed from the major and melodic minor scales and are separated by intervals of whole-tones (*tones, W*) and half-tones (*semitones, H*). Modes that are constructed from the harmonic minor scale include an interval of three semitones as well (Figure 5). Seven modes can be constructed out of each major (Figure 6), melodic and harmonic minor scales (21 modes all together).

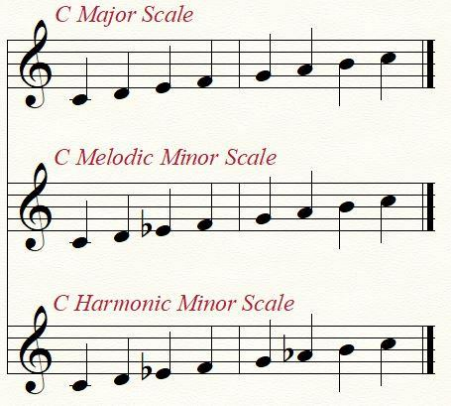


Figure 5: C major, melodic and harmonic minor scales

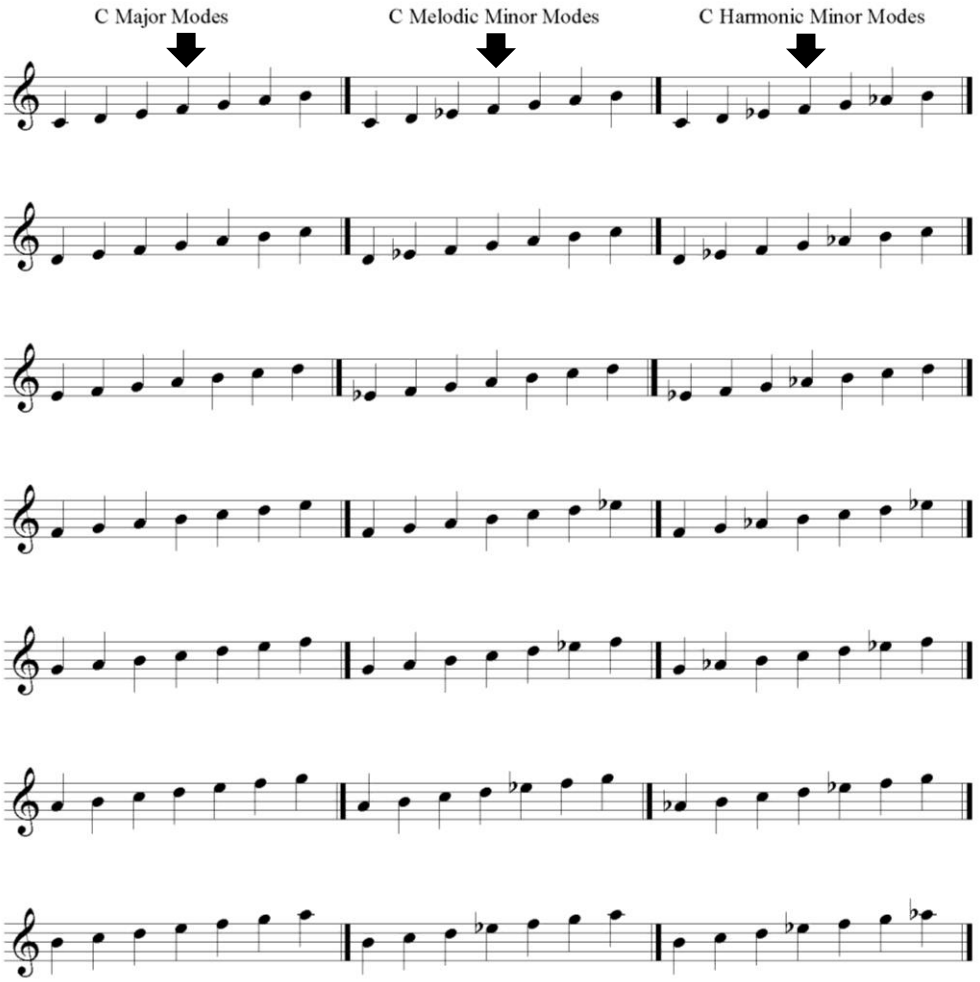


Figure 6: Modes constructed from C major, C melodic minor and C harmonic minor scales

Since the PSM algorithm analyzes a musical phrase, it is important to expand on what a musical phrase means. For that matter, the term musical phrase, as referred to in this paper, is any length of a melodic line, played in-or-out of time, rooted within one tonality that completes a sense of its own. When improvising a phrase for the algorithm to analyze that is based in one tonality, three possible scenarios can accrue: (1) The phrase contains *only* seven unique notes that are required to form a scale (Figure 7). (2) The phrase contains *more* than seven notes (Figure 8). (3) The phrase contains *fewer* than the seven notes that are required to form a scale (Figure 9).

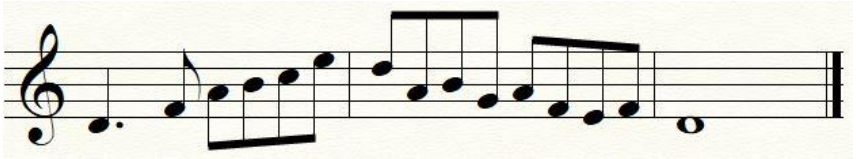


Figure 7: D Dorian diatonic phrase

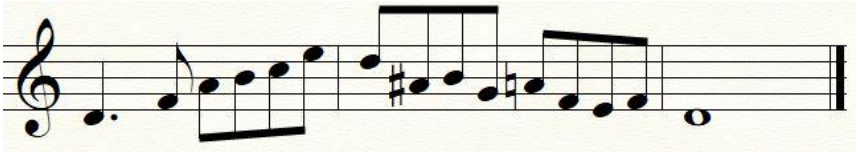


Figure 8: D Dorian phrase with additional note (A#)

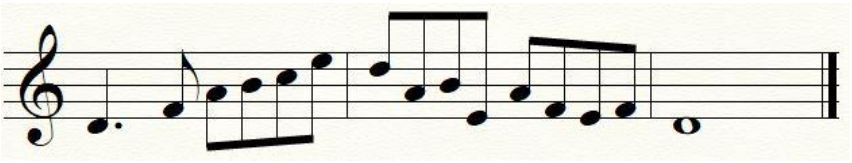


Figure 9: D Dorian phrase with a missing note (G)

There are several qualities that make for a good melodic line. A sense of direction and a climax-point are among the most important ones. Duration, dynamic level, and placement of notes within the melody will also affect the character of the outcome. When discussing the single melodic line, Kennan (1972) highlights the relative importance of notes. In a melodic line,

> certain notes are heard as being more important than others. This may occur when those notes are: (1) the highest or the lowest in a phrase or a longer segment; (2) the first and/or the last; (3) longer in value; (4) repeated, either immediately or later; (5) in a strong metric position; (6) accented dynamically; (7) harmonic as opposed to nonharmonic; (8) in a step-progression.

The PSM algorithm takes into account three of those points (2,3 and 4). In order for the PSM algorithm to work, the first note of the played phrase must be the tonic, the first degree of the scale. Secondly, the duration for each note in the phrase is collected, calculated and translated into a weight. Notes with a longer duration will be counted as more important than others. Thirdly, the number of occurrences for each note in the phrase is translated into a weight as well, where notes that have repeated more are counted as more important than notes that repeated less frequently or that were not present.

While modes are characterized first and foremost by their 1st degree (the first note), they all have one and even sometimes two additional characteristic notes, which correspond to an interval that only they have.[15] The natural modes of the major scale distinguish themselves due to their natural characteristic degrees (Appendix A). Altered modes from the melodic and harmonic minor scales distinguish themselves by their altered characteristic degrees (Appendix A). The PSM algorithm is provided with the natural (in major) and altered (in minor) characteristic degrees for each of the 21 modes (referred to as indicator notes, Table 1). An adjustable weight can be defined to decide the impact of the indicator notes when calculating a scale rank.

[15] Characteristic notes in the modal system - Audiofanzine - https://en.audiofanzine.com/music-theory/editorial/articles/characteristic-notes-in-the-modal-system.html

Major Scales	1st Degree	2nd Degree	3rd Degree	4th Degree	5th Degree	6th Degree	7th Degree
Mode Name	**Ionian**	**Dorian**	**Phrygian**	**Lydian**	**Mixolydian**	**Aeolian**	**Locrian**
Intervals	2212221	2122212	1222122	2221221	2212212	2122122	1221222
Pitch Classes	02457911	02357910	01357810	02467911	02457910	02357810	01356810
Indicator Notes	4 5 11	3 9 10	1 3 (8 10)	4 6	4 9 10	3 8	1 3 6
Minor Melodic							
Mode Name	**Melodic Minor**	**Dorian b2**	**Lydian Aug**	**Mixolydian #11**	**Mixolydian b6**	**Locrian Nat 9**	**Altered Dominant**
Intervals	2122221	1222212	2222121	2221212	2212122	2121222	1212222
Pitch Classes	02357911	01357910	02468911	02467910	02457810	02356810	01346810
Indicator Notes	3	1 3 10	4 6 8	4 6 10	4 8 10	2 3 6 8 10	1 3 4 (6 8 10)
Minor Harmonic							
Mode Name	**Harmonic Minor**	**Locrian Nat 6**	**Ionian Aug**	**Dorian #11**	**Phrygian Major**	**Lydian #9**	**Altered Dominant bb7**
Intervals	2122131	1221312	2213121	2131212	1312122	3121221	1212213
Pitch Classes	02357811	01356910	02458911	02367910	01457810	03467911	0134689
Indicator Notes	3 8 11	1 3 9	4 8	3 6 10	1 4 8 10	3 4 6	1 3 4 8 9

Table 1: Modes of the major, melodic and harmonic minor scales, including interval, pitch-class and indicator notes

To summarize, several music theory concepts are being implemented within the PSM algorithm. Some require no real calculation, and some are calculated as weights to impact the output:

1. The first note of the phrase is the tonic – a set condition, no calculation is being done
2. Notes that repeat more in the phrase are viewed as more important
3. Notes that are played longer are viewed as more important
4. Characteristic notes are considered more important.

2.3.2 Model analysis object

The '*Modal Object Library'* developed by Manzo (2007) is an open-source collection of algorithms[16] meant to define and control modality in the Max/MSP/Jitter programming environment. The library was created to aid the author with his "own compositional interests including algorithmic composition and interactive music systems." Available in that library is the '*Modal Analysis Object*' (Figure 10). It takes incoming notes (melody or phrase) and determines the mode and tonic of the melody by attempting to filter out repetitions and organize the notes. The object provides good analysis when the melody is exclusively constructed by the notes of the mode, however, it lacks any consideration for notes that are outside of the mode (chromaticism). Also, the modal analysis object filters out note repetition and does not care for notes' duration. In contrast, the PSM algorithm utilizes note repetition and notes duration to help determine the mode of a phrase. Manzo (2007) attests that the "Modal Analysis (object) has some shortcomings" because it considers the lowest pitch of the played phrase as the scale tonic (first degree). The PSM, in a way, solves this problem by setting the condition that the first note of the played phrase is the tonic of the scale. After playing the first note and by that, setting the tonic, the player may play in any range without affecting the set tonic.

[16] EAMIR - http://www.eamir.org/

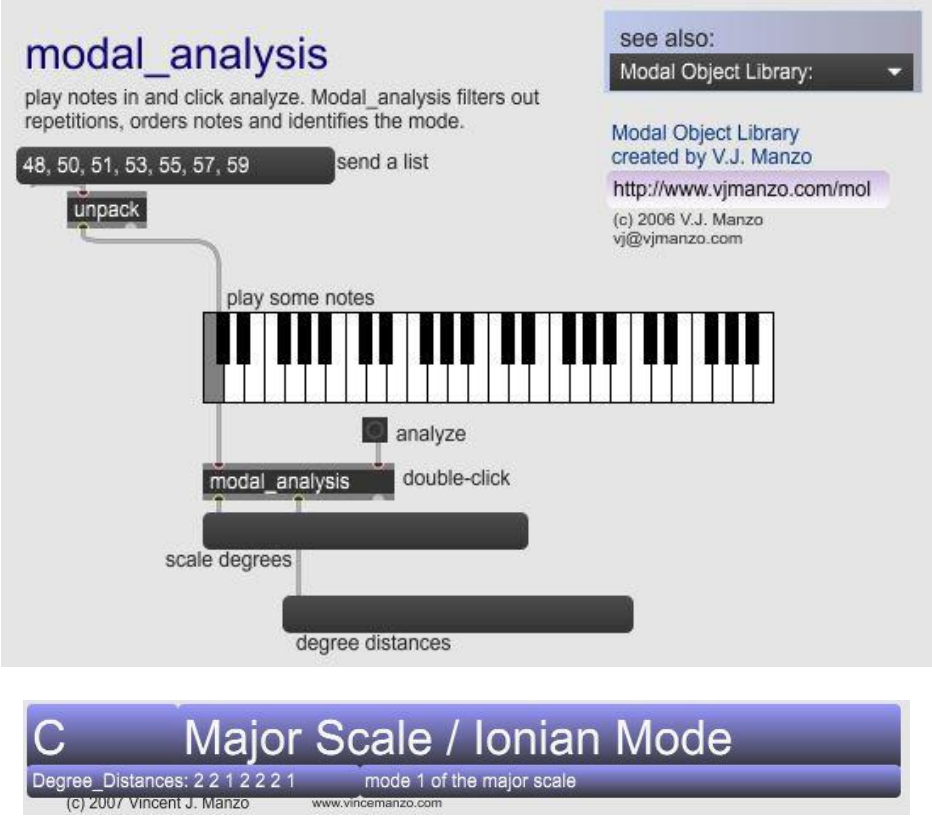


Figure 10: Modal Analysis object by V.J. Manzo

2.3.3 Music Information Retrieval (MIR)

MIR, the interdisciplinary science of retrieving information from music, carries much of the research in the field of music analysis. Many MIR tasks and methods are being used by businesses and academics to categorize, manipulate, and create music. Music identification, plagiarism detection, and copyright monitoring are just some of the tasks being used by businesses regularly (Casey *et al.*, 2008). The MIR field uses both symbolic and audio data sources to perform analysis by using approaches like metadata, and extraction of high-level and low-level audio features. Low-level audio features are measurements of audio signals that contain information about a musical work and music performance. Low-level audio features are segmented in three different ways: frame-based segmentations, beat-synchronous segmentations, and statistical measures that construct probability distributions out of features. Numerous low-level audio features are based on the short-time spectrum of the audio signal. Among the many methods used are: short-time magnitude spectrum, Constant-Q/Mel spectrum, onset detection, and Mel/Log-Frequency cepstral coefficients (Casey *et al.*, 2008). Another method for low-level feature extraction that can help with identifying the key and mode (high-level features) of a given melody is the pitch-class profile or PCP (Chromagram). In this method, the octave is divided into 12 equally spaced pitch classes (the 12 notes of Western tonal music). This feature integrates the energy in all octaves of one pitch class into a single band. The result can be converted into a pitch-histogram and later analyzed and compared against a scale database to identify the key and mode (high-level music content

description) (Casey *et al.*, 2008). The PSM algorithm works in the same way, in that it folds all the notes of a phrase into 12 pitch-classes and performs a comparison with a pre-defined scale database, while also taking into consideration the duration and number of occurrences of each pitch-class. Several chords and key recognition systems in MIR use the Hidden Markov model (HMM) to unify recognition and smoothing into a single probabilistic framework, other systems utilize estimation of melody and bass lines to identify tonality and harmony. These methods are mostly meant for research and are not designed to perform analysis and retrieve information for real-time interactive music systems.

2.3.4 Other algorithms for scale recognition

The first algorithm for detecting tonality was developed by Longuet-Higgins and Steedman (1987). The algorithm compares the tones of the musical input with the tonal region of each of the major and minor keys. Based on that idea, the Krumhansl-Schmuckler's algorithm was developed (Temperley, 1999). The algorithm correlates the distribution of pitch-class weighted according to the duration with the 24 profiles of the major and minor scales. Coming to solve the limitation of only detecting major and minor scales, Zhao (2016) has proposed "an algorithm to identify the musical scale of a monophonic melodic without modulation, which is composed of any intervallic structure and not necessarily the structure of the most popular scales." Without the need of prior knowledge of the profile scale, the algorithm can identify any scale of the 12-tone system with structure intervals of 1, 2 and 3 semitones. This is done by proposing a scale encoding system where "each scale is identified in a unique way with a numeric vector," returning a scale code and not a scale name. Zhao's method consists of four steps: pitch selection in symbolic data, representation and encoding of musical scales, a deterministic walk through the intervallic structure digraph and a scale code calculation of the last node visited. "The scale detector performs a deterministic walk through the nodes of a predefined graph (Figure 11). In this graph, each node is an interval structure and the edges represent the possible transformations that may have an intervallic structure when its intervals are fractionated. The walk between nodes is determined by a validation rule, which determines whether adding a new interval corresponds to a correct structure."

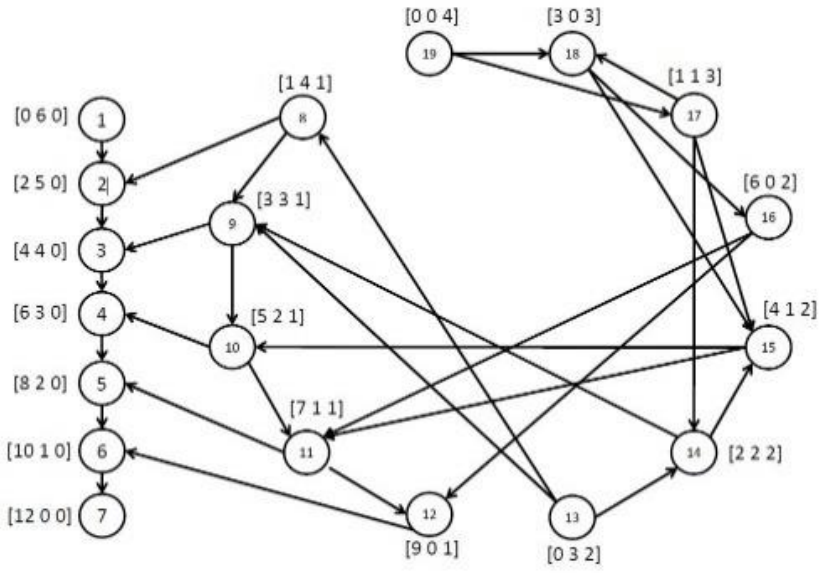


Figure 11: Zhao's graph of valid structures

This method showed high accuracy results when checked against a database of Finnish melodies composed of a full scale, and was able to detect both known and unknown scales. For melodies with an incomplete scale structure, Zhao reports that "the algorithm made a good estimate of the scale by measuring the percentage of harmonic similarity." Regarding an additional database built of random melodies composed of both known and unknown scales, the author attests that although the method showed high accuracy, "the main difficulty is to identify the mode and tonic, due to the fact that [the] melodies do not follow a method of modal or tonal composition." Zhao concludes that in order to correctly identify a musical scale, it is also necessary to consider the type of composition in the original melody.

2.4 Summary, motivation and research question

In jazz as well as other forms of improvised music, the collective's familiarity with musical concepts like rhythm, harmony, or groove provides the basis for the communication between the musicians while playing together. The exchange of information and ideas is happening at a very fast pace, and the quality of the performance is directly affected by the knowledge and skills of the individual musicians. Approaching this project, I was interested in conveying harmonic ideas by playing saxophone to a computer just like I would if a piano player was playing by my side. The first piece of information I wanted the computer to get from me was the tonality I was playing in. From the viewpoint of a saxophone player, being able only to produce one note at a time without harmonic context, my wish was to control harmony by the melodic output of the saxophone. I have searched for commercial and non-commercial applications that allow for this kind of exchange to happen in real-time for generating an expressive performance. After reviewing the field, I have decided to develop a system that will facilitate doing just that. Following advances by Robert Rowe

(2004), I was interested in designing an interactive system that would have "musical skills," and that would be capable of analyzing and recognizing musical concepts, with the focus on harmony.

The large number of electronic wind instruments, hybrid saxophones, and various augmentation projects intended to expand the output of the instrument revealed a fair amount of interest in the subject. Several of the above-discussed saxophone augmentation projects, like the HypeSax and the EMEO, present some new and exciting concepts in terms of instrument expansion (sound hybridization, connectivity); yet, almost all presented projects had to compromise and sacrifice something (i.e., the tactile interface of the saxophone, or acoustical sound). The majority of the projects utilize mapped sensory data to directly control digital signal processing, effects parameters, multimedia works, looping, or re-sampling. I have found little evidence or interest in affording the saxophone player with controlling harmony.

Allowing the computer to extract any information from playing the saxophone required developing an application for real-time pitch-tracking. The yin~ object, which is a representation of the YIN algorithm, has been chosen for the task. Available for free as an abstraction object for the Max/MSP environment, and designed for real-time interaction by one of the world's leading institutes for research in acoustics and music (IRCAM), going with the yin~ seem to be an appropriate choice.

Once the computer can track the pitch, a program needed to be developed to collect, preprocess, and analyze this information. The programing environments of choice are Max[17] by Cycling '74 and JavaScript. The *Model Object Library* by V. J. Manzo (2007), together with his practical guide to developing interactive music systems in Max (Manzo, 2016), served as a technical textbook as well as general inspiration. Theoretical musical concepts are easy to implement within the Max environment and will later be discussed. An algorithm for scale recognition had to be developed for the computer to analyze the notes of a phrase and determine the scale. Later in the chapter, I discussed several existing scale recognition-methods and algorithms that are applied in MIR research.

In conclusion, a complete system composed of a pitch detection module, and an algorithm for scale recognition needed to be developed for real-time interaction between a computer and a saxophone player improvising a musical phrase. This kind of system would enhance the saxophone by allowing for harmony manipulation via *indirect acquisition*, a term borrowed from gestural instruments data acquisition, meaning, extracting information from the acoustic signal for interactive electroacoustic music performance (Traube, Depalle and Wanderley, 2003).

The lack of a comprehensive music system designed for live performance has led me to this research question: Can a system and an algorithm be developed to successfully identify the scale

[17] Cycling '74 - https://cycling74.com/

of a musical phrase for collective music generation? Furthermore, aim to explore and present practical ways in which the system can be used to promote creativity.

3. System Description

This chapter presents an interactive music system allowing a saxophone player to improvise a phrase, track and analyze the notes, identify the scale, using the outcomes in applications for music generation and interaction. The system was developed in Max/MSP/Jitter environment (commonly referred to as Max), a visual programming language for music and multimedia by software company Cycling '74. Max was chosen due to its flexibility for creating interactive music systems, the capability to generate audio and the ability to integrate other programming languages within its environment. The PSM algorithm part of the system is written in the JavaScript programming language since the procedural operations required to identify the scale are too difficult to implement using Max objects by themselves. Figure 12 shows the different parts and processes that make up the system.

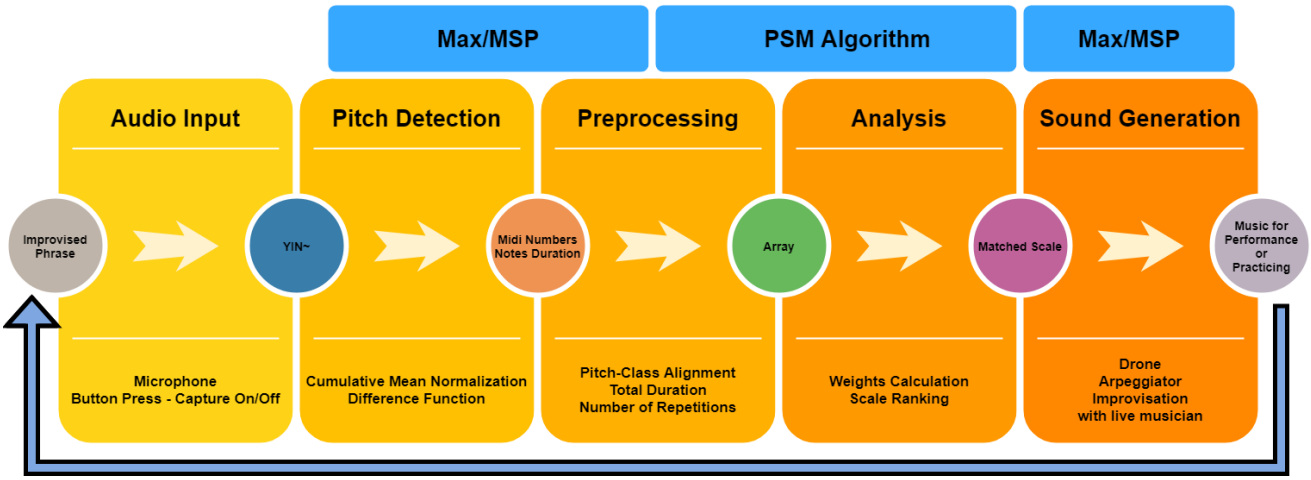


Figure 12: Diagram of the system

3.1 Audio Input

The *Audio I/O Module* consists of five user interface objects. (1) *Audio output On/Off button*. (2) *Audio Input Menu* where *Live Mode* is for performing live with the system while *Sample Mode* is designed to automatically play through pre-recorded samples for evaluating the PSM algorithm. (3) *Level indicator* to visually monitor the signal level. (4) *Monitor On/Off* button to be able to mute the system. (5) *Gain dial* for adjusting the input level.

For evaluating the system in *Sample Mode*, a database of audio recordings has been collected from five saxophonists (see chapter 4). The audio clips in the database were recorded by the players themselves with dynamic or condenser microphones or with a Zoom Handy recorder, in a quiet environment to increase accuracy when evaluating the PSM algorithm. Since using the system in a live mode means that the environment will be noisy, two types of microphones were used with the system. A condenser microphone with low gain was used for pitch tracking, and a dynamic microphone was used for recording the acoustic sound of the saxophone.

3.2 Pitch Detection

The audio signal is being sent to the *Pitch Detection Module* for pitch tracking and note detection. The player can decide when the system will start and stop capturing the notes for processing by pressing the *Capture On/Off* button. Only notes that are being played when the capture button is ON will be collected and sent to the algorithm for analysis.

3.2.1 Tuning the yin~ object

The yin~ Max object (as discussed in subsection 2.2.5) has several parameters that can be fine-tuned to achieve a better estimation of the pitch. When using the system in Sample Mode for evaluating the algorithm, no computational power needs to be reserved since the system is only employed to estimate the pitch and analyze the phrase; therefore, downsampling has been turned off (0). The lower limit of the yin estimator frequency search range is set by using a conversion from MIDI to frequency. We set the minimum frequency by choosing the lowest note available on each saxophone in MIDI notes (Table 2).

Instrument	MIDI Note Number	Note Name	Frequency (Equal tuning at 440 Hz)
Soprano Saxophone	56	G#2	207.65
Alto Saxophone	49	C#2	138.59
Tenor Saxophone	44	G#1	103.83
Baritone Saxophone (Low Bb)	37	C#1	69.30

Table 2: Saxophone's lowest note

The output period, which sets how often the yin~ object will be updating its output values, has been set to 1 millisecond. The three values coming out of the yin~ object (estimated pitch in Hz, signal amplitude, estimation quality factor) are then being sent to another abstraction object called OMax.Yin+core.

3.2.2 OMax and the OMax.Yin+core object tuning

OMax[18] (together with MoMax and WoMax[19]) is a computer-based improviser that analyses and recombines an instrumental sequence to play in real-time as a new musical partner. Designed by the Musical Representation team[20] at IRCAM, OMax combines two computer music environments (Max/MSP and OpenMusic[21]) to learn style features and co-improvise with a musician. The OMax virtual improviser Max agent is offered as a free library for research and creation.[22] The OMax.Yin+core abstraction (available from the OMax library) includes the bc.yinstats[23] external

[18] OMax | Ircam Forum - https://forum.ircam.fr/projects/detail/omax/
[19] WoMax Project Homepage - http://recherche.ircam.fr/equipes/repmus/WoMax/
[20] Music Representations Team - http://repmus.ircam.fr/
[21] OpenMusic -Music Representations Team - http://repmus.ircam.fr/openmusic/home
[22] This website | Ircam Forumhttps://forum.ircam.fr/article/detail/new-website/
[23] WoMax yinstats - http://recherche.ircam.fr/equipes/repmus/WoMax/doc/html/structt__bc__yinstats.html#_details

that implements a statistical analysis of the raw output of pitch from the yin~ object by examining all the different pitches over a time window and outputs the more probable pitch.

The OMax.Yin+core abstraction lets us adjust three crucial parameters for successful note detection. *NoiseThresh* defines a quality factor under which the input signal is rejected from the pitch detection, similar to a noise threshold. *Window (ms)* defines a time window after which an onset is validated if the estimated pitch remains stable during that time. *Consistency* directly relates to the *estimation quality factor* coming from the yin~, meaning, pitches with estimated quality under the value given will be ignored (Lévy, 2004).

When evaluating the algorithm in Sample Mode against the database, the parameters were set as such: *NoiseThresh* (NT=0.7) *Window* (W=50) *Consistency* (C=0.8). This setup proved to be accurate and consistent enough that no additional tuning was required. The failed predictions by the PSM algorithm depended mostly on the musician's level of playing and the modes being played, inaccurate indicator notes, weights adjustments within the algorithm or because of phrases containing less than seven unique notes. Additional work regarding the eventual setup for live performance and evaluation, see section 5.1.

Two outputs from the OMax.Yin+core object are used: from the left-most outlet is an *Activity* button for indicating when played notes are being tracked (button turns yellow), and from the right-most outlet are messages of pitch and velocity. The pitch/velocity messages are sent to a UI object to display the notes on a staff line together with their respected velocity presented on a slide. Having a notation representation of the notes being played can help the musician with identifying any false notes detected by the system.

3.2.3 Calculating Note Duration and Array Output

In the *calculateDuration* subpatch, the pitch/velocity messages are used to trigger a duration calculator. A high-resolution timer based on the CPU of the computer (cpuclock object) outputs the duration of each played pitch as floating-point numbers in milliseconds. The timer is only triggered by the initial velocity attack of each note, and it stops once a new or 0 velocities are received.

Each detected pitch (as midi integers) is added into a coll file together with its corresponding duration. Pressing the *Capture Off* button will send the list (stored in the coll file) into the JavaScript object in the form of an array where each data point alternates between the note played and its duration.

3.3 The Phrase to Scale Match (PSM) algorithm

The proposed algorithm takes an improvised monophonic musical phrase and matches it to a musical scale from a scale-dataset of 21 scales. For the algorithm to successfully identify the scale to which the improvised phrase belongs, two conditions must be met. First, the first note of the

musical phrase must begin with the bass note (1st degree) of the scale. Second, the improvised phrase must be based on one scale only.

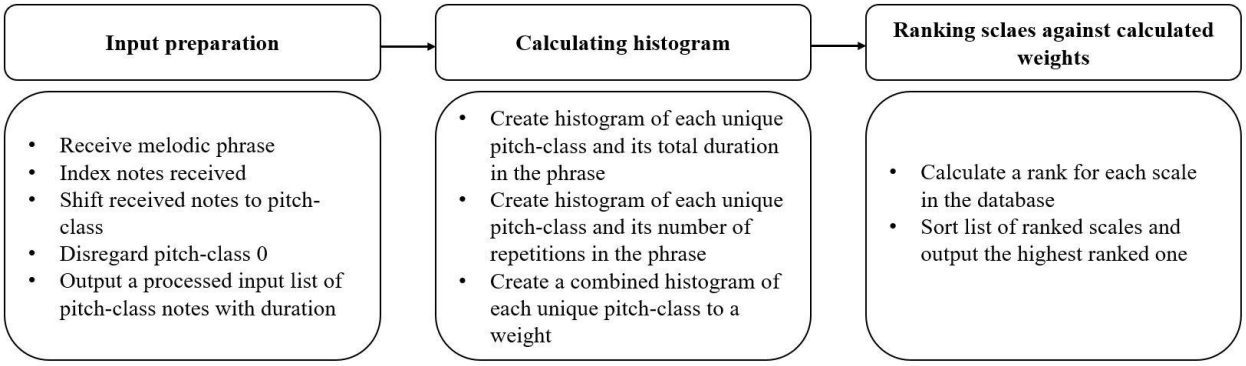


Figure 13: The three parts of the PSM algorithm

The PSM algorithm is comprised of 3 parts, as illustrated in Figure 13:

1. Aligning the input notes, so that they are all in the same octave, and are relative to the first degree.
2. Generating a histogram of the aligned notes played in the phrase, where each note is assigned a weight indicating its importance and impact.
3. Comparing the histogram against a scale-dataset, calculating a rank for each scale, and selecting the one presenting the highest-ranked score.

3.3.1 Preprocessing

The input of the preprocessing block consist of the notes played in the phrase (as midi-integers) and their respected duration (Table 3).

Playing order of notes	Midi notes	Duration
1	62	600.30
2	66	96.055
3	68	48.92
4	76	54.57
5	59	135.45

Table 3: Example of the input of the preprocessing block

The data is received as an array, with each data point alternating between the note played and its duration. Example of an array (based on Table 3): [62, 600.30, 66, 96.055, 68, 48.92, 76, 54.57, 59, 135.45]. For each phrase-note in the input phrase *p*, we calculate the corresponding pitch-class note n_{pc}, relative to the base note n_b. Pitch-class, *pc*, is the numerical equivalent of a pitch regardless of the octave (Figure 14). By using modular arithmetic (Modulo 12), each midi note

integer is divided by 12. The remainder of this division is pitch-class (equation 6). For example, all C's, regardless of octave designation, are equal to 0; all C#'s are equal to 1; all D's equal 2, and so on. Pitch classes are numbered 0–11. The number 12 is a C, so it is regarded as pitch class 0 (Manzo, 2016).

$$X \; mod \; 12 = pc \tag{6}$$

Equation 1: Modulo 12 equation

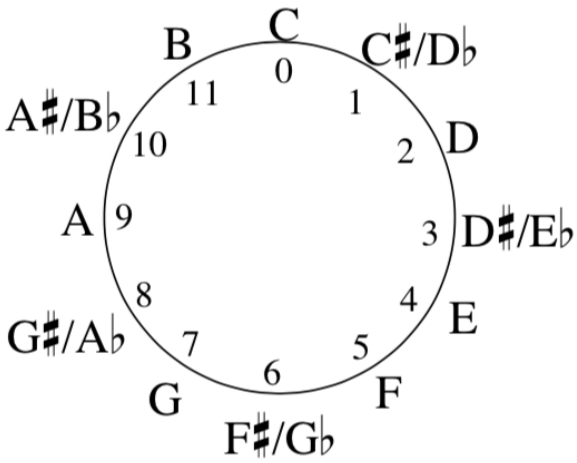


Figure 14: Pitch-Classes presented in a circle [24]

Each of the *n* notes in the phrase *p* is denominated with a 0-based index, *i*. This means the first note in the phrase is always p_0, the second p_1, the third p_2, through p_{n-1}. In this preprocessing phase, we run through the phrase notes – skipping the first, bass note – and generate a new set, *s*, which holds the corresponding pitch-class value *sk* for each *pi*, where *k=i-1*.

In some cases, the bass note n_b is not necessarily the *lowest* note in the input, and p_i - n_b might result in a negative value. Running each p_i through the "positive modulo" calculation, as seen below, guarantees that the result of each n_{pc} is in the range [0, 11]. Since the 1st degree (p_0), sometimes referred to as bass note n_b, is a member in each scale by definition, it is removed from any processing and calculations. The aligning algorithm described in pseudocode is presented in Figure 15:

```
n_b = p_0   (bass note is indexed as the first note of the phrase)
s = ∅       (represents an empty set)
for i = 1 to n-1 do
      n_pc = (((p_i - n_b) % 12) + 12) % 12
      s_i-1 = n_pc
```

[24] Image taken from https://fundamentalsofmusictheory.umasscreate.net/unit-8/

Figure 14: Positive modulo preprocessing block in pseudocode

Table 4 displays an example run through the positive modulo for a given input set, *p*, resulting in the output set, *s*:

p	67	62	57	62	58	70	64
i - index	0	1	2	3	4	5	6
s	skipping *i=0*	7	2	7	3	3	9
k = i-1		0	1	2	3	4	5

Table 4: Example of an input set and the result after running the positive modulo algorithm

3.3.2 Calculating histograms

The next step is to assign each unique pitch-class note n_{pc} with a value indicating its weight in the system – the higher the weight, the more impact that note will have on the ranking of scales that include that note. We first create two separate histograms:

1. h_{td} - a histogram accounting for the total duration n_{td} of the pitch-class note n_{pc}.
2. h_{nr} - a histogram accounting for the number of repetitions n_r of the pitch-class note n_{pc}.

We assign a linear impact for the total duration, n_{td} - meaning, a note that was played twice as long as a different note has double the impact on the weight given to it. As for the number of note repetitions, n_r, we assign an exponentially increasing weight to notes that were played multiple times in the input phrase. Setting an exponential increase helps diminish the impact for seldomly played notes, while promoting the system to give a more prominent note weight, n_w, to notes that were played multiple times. For example, setting a repetition factor f_r to 1.2 will provide a sufficient increase in weight, without skewing the results too quickly, as can be seen in Table 5:

Number of repetitions	Weight impact $f_r^{n_r}$
1	1.200
2	1.440
3	1.728
4	2.073
5	2.488

Table 5: Weight impact by the number of repetitions

For associating a single note weight n_w to each pitch-class note n_{pc}, the two histograms are merged by using the equation (7) below. The weighted value assigned to each note, n_w, equals that note's

total duration n_{td}, multiplied by the repetition factor, f_r, by the power of that note's number of repetitions, n_r, in the input phrase.

$$n_w = n_{td} * f_r^{n_r} \quad (7)$$

The rationale behind combining the n_{td} and n_r into a single note weight n_w is that in a melodic phrase, notes that repeat more often and are played longer, are usually considered more important than others (as discussed in subchapter 2.3.1 - General and related music theory concepts). Having a single weight combining those two factors helps simplify the process of determining the relative importance of notes. The result is a weighted histogram h_w of pitch-class notes n_{pc} and their corresponding note weights n_w.

3.3.3 Ranking scales against the calculated weights

For identifying the scale of the played phrase, two types of inputs are used in this block: (1) a weighted histogram h_w for each n_{pc} (already calculated in the previous block); and (2) a scales dataset s consisting of 21 scales and their indicator notes (Table 6). The decision to only use these 21 scales is based on the fact that they are the most common ones used in the majority of tonal music since the 1600s. Also, these 21 scales (7 modes of Major, Melodic Minor, and Harmonic Minor) are widely used when improvising within the Jazz genre. Each scale is represented by its name and consists of scale notes s_n and scale indicator notes s_{in}. The scale notes s_n are six unique values between 1-11, which, together with the first-degree note (always a 0), complete the seven unique notes that form that specific scale. The scale indicator notes s_{in} (a subset of s_n) are characteristic notes that are unique to each scale (previously discussed in subsection 2.3.1 – General and related music theory concepts). When they appear in a phrase, indicator notes essentially function as small hints, helping us identify one scale over another. To give a bigger impact to indicator notes s_{in} when calculating a scale rank, we introduce another factor - a linear, weight-increase factor of indicator notes f_i. This factor gives a higher weight for scale indicator notes when observed in a given phrase. Attributing higher weights to notes that are indicator notes s_{in} and to notes that repeat more often means essentially making them more important when calculating a scale rank. The example in Figure 16 illustrates the main idea behind the weight concept. The musical phrase consists of notes based in one of these two scales (Ionian or Lydian), but pitch-class note 6, which is also an indicator note of the Lydian scale, appears three times while pitch-class note 5 appears only one time. Therefore, the Lydian scale will score a higher rank than the Ionian scale.

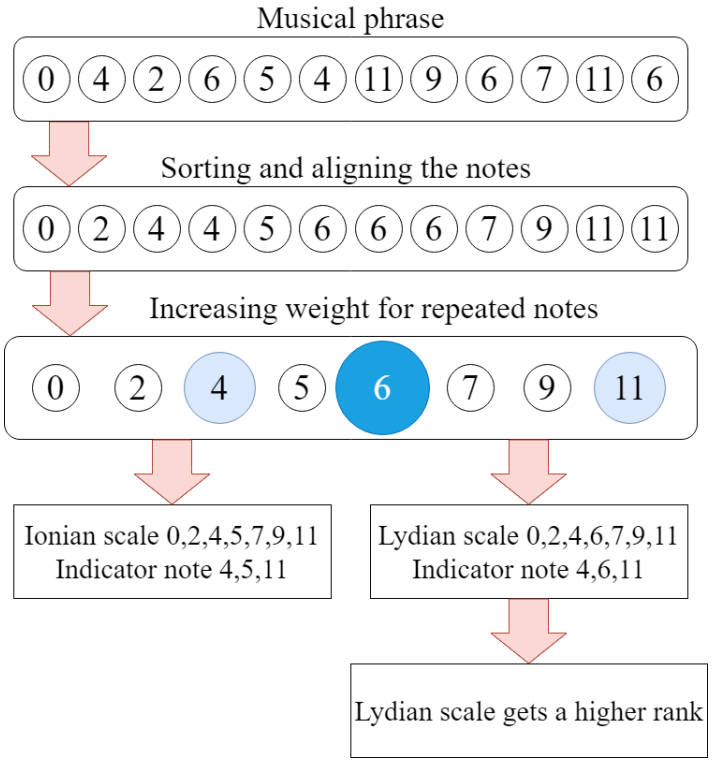


Figure 16: Appling weight-increase to repeated and indicator notes

	Scale Name	Scale Notes	Indicator Notes
Major Modes	Ionian	2,4,5,7,9,11	4,5,11
	Dorian	2,3,5,7,9,10	3,9,10
	Phrygian	1,3,5,7,8,10	1,3,10
	Lydian	2,4,6,7,9,11	4,6,11
	Mixolydian	2,4,5,7,9,10	4,5,10
	Aeolian	2,3,5,7,8,10	3,8,10
	Locrian	1,3,5,6,8,10	1,3,6,8
Melodic Minor Modes	Melodic Minor	2,3,5,7,9,11	3,11
	Dorian b2	1,3,5,7,9,10	1,3,9
	Lydian Aug	2,4,6,8,9,11	4,6,8
	Mixolydian #11	2,4,6,7,9,10	4,6,10
	Mixolydian b6	2,4,5,7,8,10	4,8,10
	Locrian Natural 9	2,3,5,6,8,10	2,3,6,8
	Altered Dominant	1,3,4,6,8,10	1,3,4
Harmonic Minor Modes	Harmonic Minor	2,3,5,7,8,11	3,8,11
	Locrian Natural 6	1,3,5,6,9,10	1,3,9
	Ionian Aug	2,4,5,8,9,11	4,8
	Dorian #11	2,3,6,7,9,10	3,6,10
	Phrygian Major	1,4,5,7,8,10	1,4,10
	Lydian #9	3,4,6,7,9,11	3,4,6
	Altered Dominant bb7	1,3,4,6,8,9	1,3,4,8,9

Table 6: Scales dataset

We first calculate each scale's rank, s_r, and then sort the scales based on their calculated rank. The higher the rank, the more likely it is that that scale is the one played in the input phrase. After outputting the scale name representing the highest score from the JavaScript Max object, music generating applications, and interactive systems can utilize this information for music-making. The pseudo-algorithm for this portion can be described as such (Figure 17):

```
for i = 1 to size(s)  (for each scale in the scales-dataset)
      r = 0  (holds the rank for the current scale, Si)
      for k = 1 to size(si)  (run through the scale's notes)
            n = sik  (holds a note in the current scale)
            if n∈hrn:  (if that scale's note appeared in the input phrase, then:)
                  nw = hw,n  (use the note's calculated weight)
                  if n∈sin  (additionally, if the note is an indicator note)
                        nw = nw * fi  (increase weight by indicator factor)
                  end if;
                  r = r + nw  (aggregate note weights as the scale's rank)
            end if;
      Ri = r  (save the rank for scale Si in the ranks set R)
end loop;
R = sort(R)  (sort the scales by their calculated ranks)
return r0  (return the top-ranked scale)
```

Figure 15: Calculating a scale rank

In some instances, when evaluating the database, the algorithm would mark more than one scale with the same rank. This can happen when the phrase contains *fewer* than the seven notes that are required to form a full scale (see Figure 6). The solution I offer is to output the most top scale name as they appear in Table 7. To give an example, a phrase that contains the notes 2,4,7,9,10 can be identified as both Mixolydian, the fifth mode of Major, or Mixolydian #11, the fourth mode of Melodic Minor. The solution offered is based on the idea that the first seven scales in the list are the modes of the Major scale and are the most used in popular and improvised music. Coming later in the list are the seven modes of the Melodic Minor scale, and last are the seven modes of Harmonic Minor modes. The solution is not optimal, and it can be addressed in future work. However, this offers a good compromise by allowing the system to choose a more accessible and more popular scale over difficult and less popular one, or even to not decide at all.

Input Phrase	2,4,7,9,10	
Mixolydian	2,4,5,7,9,10	4,5,10
Mixolydian #11	2,4,6,7,9,10	4,6,10

Table 7: Example of a rank tie between two scales

3.4 Game controller adaptation

A retro-like Super Nintendo Entertainment System (SNES) controller is used as the user-interface of the system, similar to the one in Figure 18 [25]. The Max *hi* object allows for obtaining data from human interface devices. The Nintendo USB gamepad features 12 momentary push buttons, and was chosen because its shape fits well on top of the low B and Bb key-guard of the saxophone, as visible in Figure 19, and also because it allows for easy access with the right-hand which is employed less than the left-hand when playing the sax. The adoption of the SNES controller is inspired by one of the principles of Cook (2009) on redesigning computer music controllers: ”funny is often much better than serious.” A Nintendo controller illustration has been added to the GUI of the system together with an overlaying buttons layer to provide visual feedback when pressing the buttons.

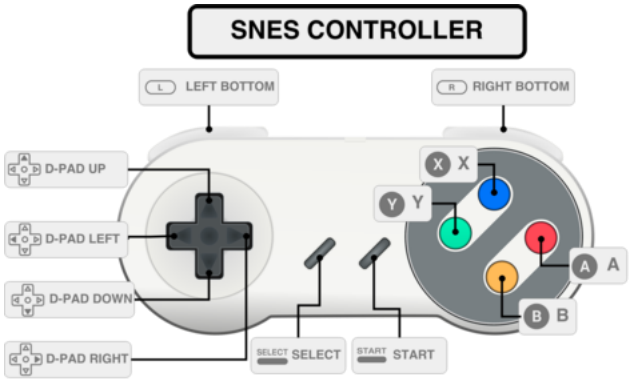


Figure 16: SNES Controller

Figure 19: The Nintendo controller attached to an alto saxophone

Since the majority of this project focuses on the pitch-detection and the PSM algorithm parts, only two fairly simple sound generating modules have been developed so far to demonstrate the

[25] Controller configs - https://www.raspberrypixtreme.com/page/controllerconfigs

system's potential. For this reason, not all of the controller's buttons are employed, and no meaningful mapping strategy has been formed.

3.5 Music Generation Modules

The JavaScript object outputs a scale name based on the analysis done by the PSM algorithm. This information can now be used in various ways for interactive music generation. At the moment, two music generating modules have been developed to give an example to the kind of opportunities the system offers.

3.5.1 Drone Module

In this module, the 1^{st}, 3^{rd}, 5^{th} and 7^{th} degrees of the output scale are extracted to form what is known as a "seventh chord". The notes of the chord are converted from pitch-class notes to midi integers based on the key of the phrase, and then frequencies. The four frequencies, representing the four notes, are played by four sinusoidal oscillators with randomized amplitude in different time points and lengths. This creates a drone effect where the chord is continuously playing while offering a changing harmonic texture. A Rotary knob allows for controlling the gain output of the Drone module. The player can continue improvising on the scale until deciding to feed the algorithm with a new harmonic environment by pressing the *Capture On/Off* button again.

3.5.2 Arpeggiator Module

The Arpeggio module is like the Drone module, but this case, all seven degrees of the output scale are converted to frequencies and are randomly played over three octaves with bell sounds. The notes are panned back and forth to play between the left and right channels (ping-pong effect); two rotary knobs allow for controlling the speed of the arpeggiator and the volume. The speed knobs of the arpeggiator and the option to set the step size of the speed buttons (by increments of 10, ranging between 10-100) are mapped to the controller's D-Pad in this way: up – faster, down – slower, right – decrease step size, left – increase step size.

A combination of the Drone and the Arpeggiator module gives a very subtle harmonic textural background for a performer to play and improvise over. This can also serve as a practicing tool to exercise scales and improvisation. By creating a randomized melody, the bell-like notes can also serve as a melodic inspiration for the improviser, though this interaction only goes one way. More examples of additional interaction opportunities this system can offer are discussed under the future work section.

4. Evaluation

4.1 Evaluation Method

The PSM algorithm can be evaluated in both live and sample mode. The benefit of using a database of recordings to evaluate the system is the assurance of accuracy and consistency compared to attempting to play the exact same phrase precisely in a live situation, for example. Another benefit is that the system and the algorithm can be monitored in real-time using log messages to identify any problems. The database can be pushed into the algorithm as many times as needed, to allow fine-tuning of the gain levels, the pitch detection parameters of the yin~ and OMax objects or the weight factors in the algorithm.

In the development phase, I have used 84 recordings (4 recordings per scale, 21 scales) of myself improvising with an alto saxophone. The results from the algorithm were not consistent across my phrases, and around 60% of the phrases were recognized correctly. By viewing the scale-ranking log, I could identify that the problem lay with the indicator notes: whereas some scales held only one indicator note, other scales held three or four indicator notes. This affected the scale ranking calculation because scales with one or two indicator notes will rank lower than scales with three or four indicators. Realizing that the indicator notes required additional adaptation, I created a scale diagram (Figure 20) that illustrates all note alterations from the Major scale (Ionian, center). Scales marked in red have a lowered 3rd degree (pitch-class 3), scales marked in yellow have a natural 3rd degree (pitch-class 4), and scales marked in blue holds both (pitch-class 3 and 4). I grouped the scales based on their notes alterations and was able to visually identify scales that might be in conflict when calculating rank. This resulted in a more balanced set of indicator notes (see scales dataset - Table 6), which also helped to increase the success rate of the algorithm from 60% to 90%.

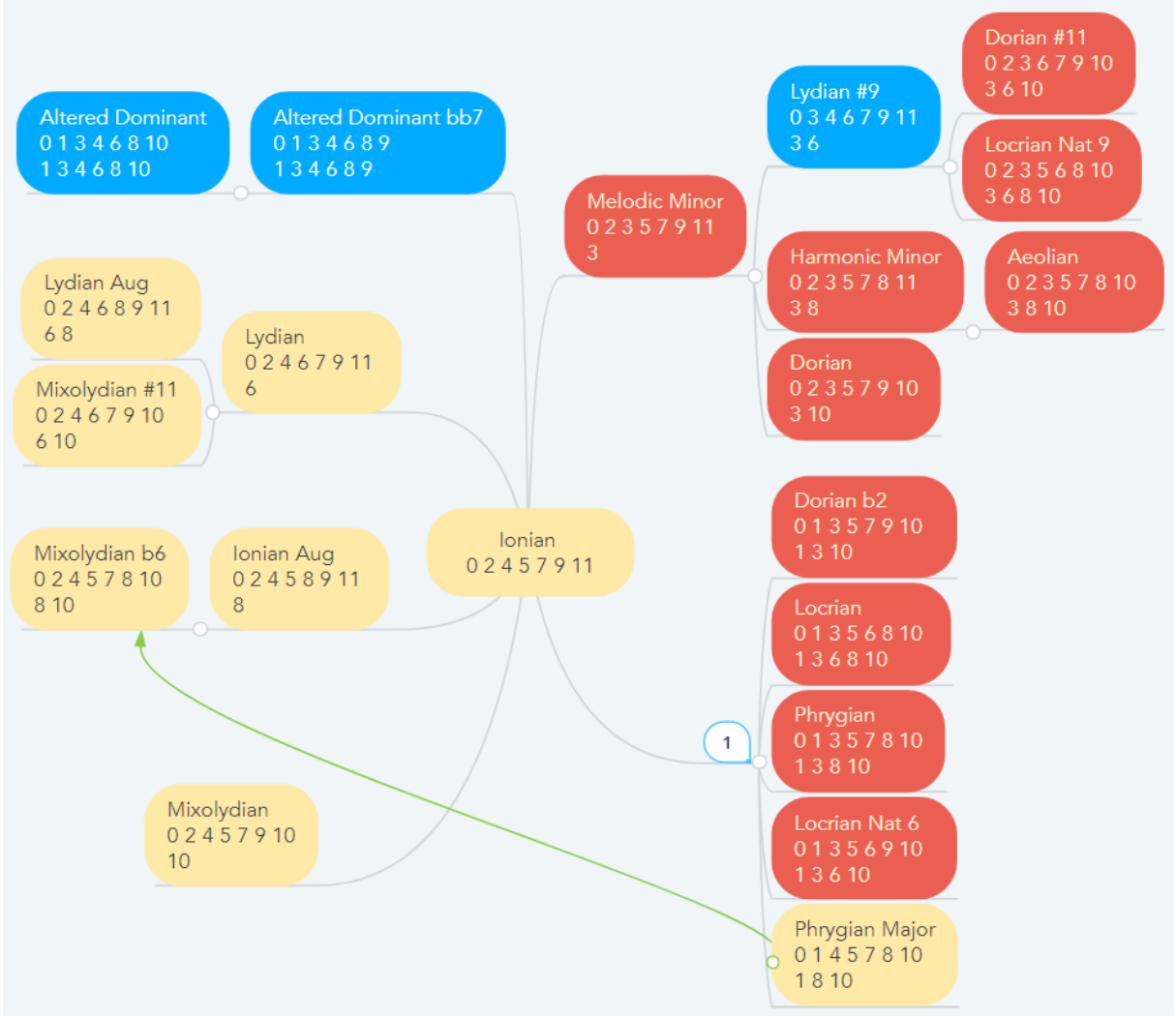


Figure 20: Scales mind map organized by name, pitch-class, and alterations of the major scale

Once the algorithm showed an improvement in the prediction rate, I expended the evaluation by inviting other saxophonists to try it as well. Having different saxophone players play with the system is crucial for the simple reason that saxophone players can be very diverse in terms of playing style, articulation, sound, improvisational vocabulary and melodic ideas. By increasing the test subjects, I essentially made sure the algorithm is not specifically designed for my own way of playing. Evaluating the algorithm with different saxophone players will help confirm that it is suitable for use by and can be beneficial to other musicians as well.

To evaluate the PSM algorithm, I gathered a database that consists of audio recordings from five saxophone players. All players are trained as jazz saxophonists and hold (more or less) the same level of music education with a focus on jazz. Three of them have graduated from the Norwegian Academy of Music (NMH), one is a Berklee College of Music graduate, and another one from the Dutch academy of music located in Amsterdam (CvA). Out of the five Saxophonists, one is a female and their ages range between 25-40. Three play the alto saxophone, two play the tenor

saxophone, and between all five saxophone players, there are three nationalities. All of the players practice improvisation in their daily lives to various degrees and were approached because I was familiar with their level and quality of playing, trusting they hold the capacity to convey a scale by playing an improvised phrase. All players have received the same explanation and guidelines for performing the task (Appendix B) as well as a scale chart (appendix C) to make sure there will not be any confusion. Some of the key instructions in my letter to the other players were: (1) The first note of the phrase must be the first degree of the scale. The reason for that is that the preprocessing block of the algorithm, including the alignment of the notes and the conversion to pitch-classes, is in reference to the first note of the phrase. (2) The phrase must be based in only one key and one scale. The algorithm can only output one key and scale per phrase and is not designed to recognize any modulation. (3) The player should try to avoid bending notes because the pitch tracker will identify them as other notes than intended. (4) It is important to correctly label the sound files with the key and scale name because the software will eventually compare between the expected scale (based on the file name) and the outcome from the algorithm. Additional points that were mentioned are: to play the phrase just as they would try to convey a key and scale to a band standing with them on stage, to provide a decent quality of audio recording and to avoid rapid phrasing for the fear that the pitch detection module would miss some of the notes and therefore would fail the scale prediction. I have also mentioned to them that a "phrase can be long or short, can contain chromaticism, and may or may not have all the notes of the scale." This remark is a vital one because I was interested in evaluating the performance of the algorithm in any of these three situations: (1) The phrase contains *only* seven unique notes that are required to form a scale. (2) The phrase contains *more* than the seven notes that form a scale. (3) The phrase contains *fewer* than the seven notes that are required to form a scale.

Each saxophonist provided me with 21 audio clips of themselves playing an improvised phrase in all modes, in various keys and playing styles (straight/swing feel, different articulations, in-tempo/rubato). Recorded in a quiet environment, the lengths of the phrases varied between 7-20 seconds. The speed of the phrases varied between the slow-and-fast-paced type of playing with different dynamic levels. Each file name was labeled by the player with the name of the scale they played in and the first note, the bass note, that they started with (for example, Ionian. F, Phrygian. Ab). Recorded with a Zoom Handy recorder or by a dynamic or condenser microphone, the audio clips were provided in mp3, wav, and aiff formats. No additional post-processing has been done to the audio files by me.

4.1.1 Limitations

The initial plan was to evaluate the whole system, and not only the algorithm, by having several saxophonists play with it. This would have included letting them use the attached controller together with the sound generation modules (Drone, Arpeggiator), but due to the COVID-19 epidemic, and the quarantine and social distancing that were forced upon us, a full evaluation of the system was an impossible task to carry out.

4.2 Evaluation Process

I created a playlist that holds all the audio clips for each saxophonist and a system that would automatically play through all samples and would trigger the capture On/Off button when a new audio clip starts and stops. The pitches are being tracked, and an array of the notes together with their duration is sent to the JavaScript max object for analysis. The JavaScript code includes a section responsible for comparing the input file name (labeled with the scale name) with the output of the algorithm. Log messages (Figure 21) with information about the analysis process and final results from the algorithm are sent to the Max console and allow for easy monitoring and troubleshooting. Log messages will show: (1) name of the file that started playing in the playlist, (2) original note input with rounded duration (only rounded for visual purpose), (3) bass note as a MIDI integer, (4) the processed input notes with duration (excluding the bass note) and aligned in reference to the bass note as pitch-classes, (5) a histogram of pitch-classes, number of repetitions, total durations and their calculated weights, (6) total amount of notes received, (7) number of unique notes received.

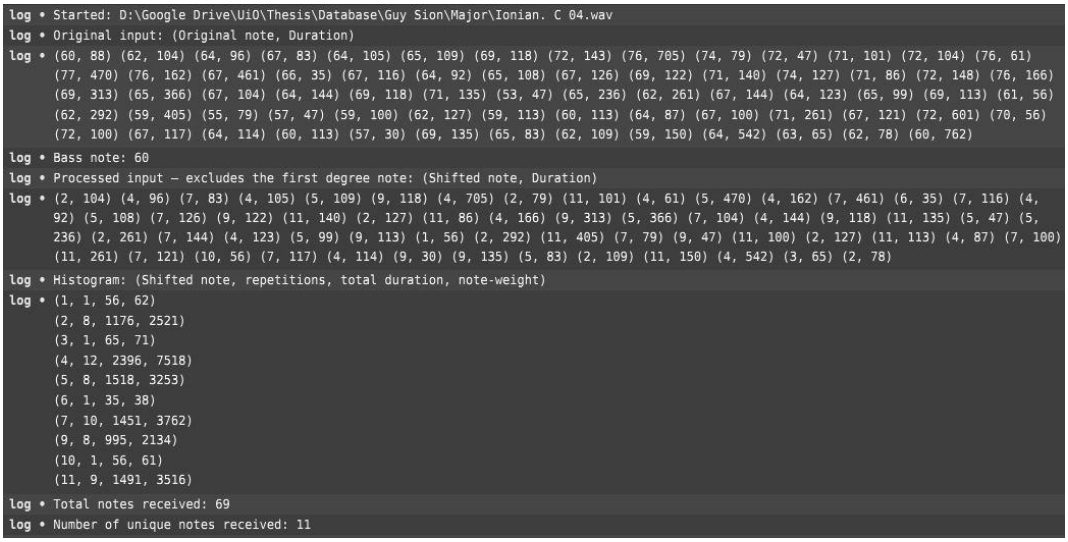


Figure 21: Example of a log message

The second part of the log file gives us a list of scales in ranking order. The example in Figure 22 shows a successful match of the algorithm: the expected scale was Ionian, and the top-ranked scale calculated by the algorithm is indeed the Ionian scale. The two last messages of the log file show what was the expected scale (based on the labeled file name) and what scale the algorithm found.

```
log • Scales by rank: (Scale name [scale notes], [indicator notes], Scale rank)
log • Ionian [2,4,5,7,9,11], [4,5,11]: 24133
log • Lydian [2,4,6,7,9,11], [4,6,11]: 20597
log • Mixolydian [2,4,5,7,9,10], [4,5,10]: 20333
log • Ionian Aug [2,4,5,8,9,11], [4,8]: 19694
log • Mixolydian b6 [2,4,5,7,8,10], [4,8,10]: 17874
log • Lydian #9 [3,4,6,7,9,11], [3,4,6]: 17803
log • Mixolydian #11 [2,4,6,7,9,10], [4,6,10]: 16797
log • Lydian Aug [2,4,6,8,9,11], [4,6,8]: 16483
log • Melodic Minor [2,3,5,7,9,11], [3,11]: 15616
log • Phrygian Major [1,4,5,7,8,10], [1,4,10]: 15421
log • Harmonic Minor [2,3,5,7,8,11], [3,8,11]: 13482
log • Dorian [2,3,5,7,9,10], [3,9,10]: 12029
log • Altered Dominant bb7 [1,3,4,6,8,9], [1,3,4,8,9]: 10802
log • Aeolian [2,3,5,7,8,10], [3,8,10]: 9682
log • Dorian b2 [1,3,5,7,9,10], [1,3,9]: 9570
log • Dorian #11 [2,3,6,7,9,10], [3,6,10]: 8605
log • Altered Dominant [1,3,4,6,8,10], [1,3,4]: 8516
log • Phrygian [1,3,5,7,8,10], [1,3,10]: 7229
log • Locrian Natural 9 [2,3,5,6,8,10], [2,3,6,8]: 6208
log • Locrian Natural 6 [1,3,5,6,9,10], [1,3,9]: 5846
log • Locrian [1,3,5,6,8,10], [1,3,6,8]: 3502
log •
log • Expected scale file name: D:\Google Drive\UiO\Thesis\Database\Guy Sion\Major\Ionian. C 04.wav
log • Expected scale: Ionian
log • Found scale: Ionian
```

Figure 22: Example of a log message showing a successful scale match

Through the testing phase, the parameters of the pitch detection module were set as such: Noise Thresh (=0.7), consistency (=0.8), Window (ms) (=50), Min Pitch (=C#2), Down Sampling (=1). As previously mentioned (see 3.3), in the JavaScript code of the algorithm, there are two variables where one can adjust the factor values of the indicator notes f_i and the number of repetitions f_r, essentially giving more weight to notes in the phrase that are marked as indicator notes and that repeat more. 105 phrases were tested against the algorithm, or 21 phrases (in 21 scales) times five players. The entire database was pushed into the algorithm a total of eight times with different sets of factor (f_i, f_r) values. The results are plotted in a graph (Figure 23), comparing the success rate of the eight tests. In test 1, the f_i and f_r values were set to 1.0, meaning no weight increase was applied. The success rate in test 1 is 84%. Tests 6 and 7 represent the highest success rate (89%) when tested with f_i =1.1 and f_r=1.1/1.15. The result of test 1 tells us the algorithm itself, without giving any increased weight to indicator notes or repetitive notes, already have a high success rate (an average of 17.6 phrases recognized correctly out of 21 per player). When reviewing the results in Figure 23, we can also establish that increasing the success rate can be achieved by setting very small f_i and f_r values. In test 5, where the increased factor values were higher in comparison to the other tests (f_i =1.25 and f_r=1.2), the success rate showed poor results (78%).

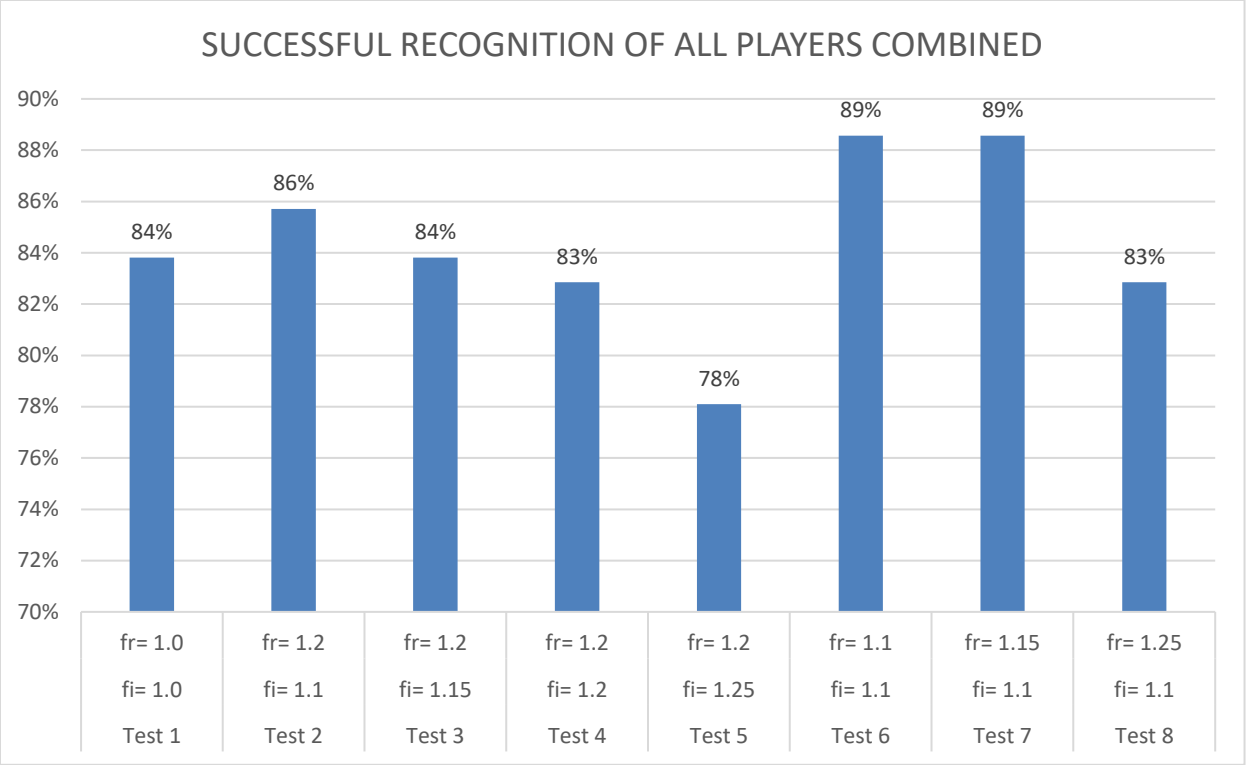


Figure 17: Comparing the success rate between eight tests

It is important to discuss several crucial points when coming to evaluate the algorithm with the current database. First, we should keep in mind that improvised musical phrases can be seen (or heard) in a very subjective way, where one player can consider a phrase in one scale while another player can consider the same phrase in a different scale. The perception of phrases and their scales can differ substantially from one player to another. The PSM algorithm is, in a way, a deterministic algorithm, attending a subjective or an individual problem. Second, tuning (of pitch) affects the algorithm's prediction. Several of the recorded phrases begin with a bent note or contain untuned notes throughout the phrase. A bent note at the beginning of a phrase means that the bass note of the phrase is being identified incorrectly by the pitch tracker, consequently swaying the phrase prediction. Phrases containing untuned notes will always be recognized falsely, and changing the weight factors will not affect their prediction outcome. As an example, in Table 8, player 1 holds a success rate of 95% in all tests, where 20 out of 21 phrases were recognized correctly. When inspecting the results, we can see that the Mixolydian scale was the one scale mistakenly recognized in all tests of player 1. This might suggest that there is a problem with the phrase itself and not necessarily with the algorithm. These kinds of database discrepancies make it difficult to evaluate the PSM algorithm and essentially mean that a larger and more accurate database is required for evaluation (See "future work" for additional reflections regarding a database).

The presented tables (7-11) display a summary of the algorithm's performance by player, with all eight tests over 21 scales. As mentioned above regarding player 1, a fault with the Mixolydian phrase sample (a bent note) was most likely the reason the algorithm did not identify the correct

scale in all tests. The results of players 2, 3, 4, and 5 show that the prediction rate was improved by increasing the weight factor when comparing test 1 to tests 6 and 7. Arguably, the least experienced player from the five subjects (player 4) demonstrated the lowest success rate in all tests, but still showed significant improvement in success rate when tested with added weights. Figure 24 presents a visual summary of Tables 8-12.

	Test 1	Test 2	Test 3	Test 4	Test 5	Test 6	Test 7	Test 8	1=Success
	fi= 1.0	fi= 1.1	fi= 1.15	fi= 1.2	fi= 1.25	fi= 1.1	fi= 1.1	fi= 1.1	0=Fail
Player 1	fr= 1.0	fr= 1.2	fr= 1.2	fr= 1.2	fr= 1.2	fr= 1.1	fr= 1.15	fr= 1.25	Success Rate by Scale
Aeolian	1	1	1	1	1	1	1	1	100 %
Dorian	1	1	1	1	1	1	1	1	100 %
Ionian	1	1	1	1	1	1	1	1	100 %
Locrian	1	1	1	1	1	1	1	1	100 %
Lydian	1	1	1	1	1	1	1	1	100 %
Mixolydian	0	0	0	0	0	0	0	0	0 %
Phrygian	1	1	1	1	1	1	1	1	100 %
Altered Dominant	1	1	1	1	1	1	1	1	100 %
Dorian b2	1	1	1	1	1	1	1	1	100 %
Locrian Natural 9	1	1	1	1	1	1	1	1	100 %
Lydian Aug	1	1	1	1	1	1	1	1	100 %
Melodic Minor	1	1	1	1	1	1	1	1	100 %
Mixolydian #11	1	1	1	1	1	1	1	1	100 %
Mixolydian b6	1	1	1	1	1	1	1	1	100 %
Altered Dominant bb7	1	1	1	1	1	1	1	1	100 %
Dorian #11	1	1	1	1	1	1	1	1	100 %
Harmonic Minor	1	1	1	1	1	1	1	1	100 %
Ionian Aug	1	1	1	1	1	1	1	1	100 %
Locrian Natural 6	1	1	1	1	1	1	1	1	100 %
Lydian #9	1	1	1	1	1	1	1	1	100 %
Phrygian Major	1	1	1	1	1	1	1	1	100 %
Scales Identified Correctly	20	20	20	20	20	20	20	20	
Total of Scales	21	21	21	21	21	21	21	21	
Success Rate by Test	95 %	95 %	95 %	95 %	95 %	95 %	95 %	95 %	

Table 8: Player 1

	Test 1	Test 2	Test 3	Test 4	Test 5	Test 6	Test 7	Test 8	1=Success
	fi= 1.0	fi= 1.1	fi= 1.15	fi= 1.2	fi= 1.25	fi= 1.1	fi= 1.1	fi= 1.1	0=Fail
Player 2	fr= 1.0	fr= 1.2	fr= 1.2	fr= 1.2	fr= 1.2	fr= 1.1	fr= 1.15	fr= 1.25	Success Rate by Scale
Aeolian	1	1	1	1	1	1	1	1	100 %
Dorian	0	0	0	0	0	0	0	0	0 %
Ionian	1	1	1	1	1	1	1	1	100 %
Locrian	1	1	1	1	1	1	1	1	100 %
Lydian	1	1	1	1	1	1	1	1	100 %
Mixolydian	1	1	1	1	1	1	1	1	100 %
Phrygian	1	1	1	1	1	1	1	1	100 %
Altered Dominant	1	1	1	1	0	1	1	1	88 %
Dorian b2	1	1	1	1	1	1	1	1	100 %
Locrian Natural 9	1	1	1	1	1	1	1	1	100 %
Lydian Aug	1	1	1	1	1	1	1	1	100 %
Melodic Minor	1	1	1	0	0	1	1	1	75 %
Mixolydian #11	1	1	1	1	1	1	1	1	100 %
Mixolydian b6	1	1	1	1	1	1	1	1	100 %
Altered Dominant bb7	1	1	1	1	1	1	1	1	100 %
Dorian #11	1	0	0	0	0	1	1	0	38 %
Harmonic Minor	1	1	1	1	1	1	1	1	100 %
Ionian Aug	1	1	1	1	1	1	1	1	100 %
Locrian Natural 6	0	1	1	1	1	1	1	1	88 %
Lydian #9	1	1	1	1	0	1	1	0	75 %
Phrygian Major	0	1	1	1	0	1	1	1	75 %
Scales Identified Correctly	18	19	19	18	15	20	20	18	
Total of Scales	21	21	21	21	21	21	21	21	
Success Rate by Test	86 %	90 %	90 %	86 %	71 %	95 %	95 %	86 %	

Table 9: Player 2

	Test 1	Test 2	Test 3	Test 4	Test 5	Test 6	Test 7	Test 8	1=Success
	fi= 1.0	fi= 1.1	fi= 1.15	fi= 1.2	fi= 1.25	fi= 1.1	fi= 1.1	fi= 1.1	0=Fail
Player 3	fr= 1.0	fr= 1.2	fr= 1.2	fr= 1.2	fr= 1.2	fr= 1.1	fr= 1.15	fr= 1.25	Success Rate by Scale
Aeolian	1	1	1	1	1	1	1	1	100 %
Dorian	1	1	1	1	1	1	1	1	100 %
Ionian	1	1	1	1	1	1	1	1	100 %
Locrian	1	1	1	1	1	1	1	1	100 %
Lydian	0	1	1	1	1	1	1	1	88 %
Mixolydian	1	1	1	1	1	1	1	1	100 %
Phrygian	1	1	1	1	1	1	1	1	100 %
Altered Dominant	1	1	1	1	1	1	1	1	100 %
Dorian b2	1	1	1	1	1	1	1	1	100 %
Locrian Natural 9	1	0	0	0	0	1	1	0	38 %
Lydian Aug	1	1	1	1	1	1	1	1	100 %
Melodic Minor	1	1	1	1	1	1	1	1	100 %
Mixolydian #11	1	1	1	1	1	1	1	1	100 %
Mixolydian b6	1	1	1	1	1	1	1	1	100 %
Altered Dominant bb7	1	1	1	1	1	1	1	1	100 %
Dorian #11	1	1	1	1	1	1	1	1	100 %
Harmonic Minor	1	1	1	1	1	1	1	1	100 %
Ionian Aug	0	0	0	0	0	0	0	0	0 %
Locrian Natural 6	0	0	0	0	0	0	0	0	0 %
Lydian #9	0	0	0	0	0	0	0	0	0 %
Phrygian Major	1	1	1	1	1	1	1	1	100 %
Scales Identified Correctly	17	17	17	17	17	18	18	17	
Total of Scales	21	21	21	21	21	21	21	21	
Success Rate by Test	81 %	81 %	81 %	81 %	81 %	86 %	86 %	81 %	

Table 10: Player 3

	Test 1	Test 2	Test 3	Test 4	Test 5	Test 6	Test 7	Test 8	1=Success
	fi= 1.0	fi= 1.1	fi= 1.15	fi= 1.2	fi= 1.25	fi= 1.1	fi= 1.1	fi= 1.1	0=Fail
Player 4	fr= 1.0	fr= 1.2	fr= 1.2	fr= 1.2	fr= 1.2	fr= 1.1	fr= 1.15	fr= 1.25	Success Rate by Scale
Aeolian	1	1	1	1	1	1	1	1	100 %
Dorian	0	1	1	1	1	1	1	1	88 %
Ionian	1	1	1	1	1	1	1	1	100 %
Locrian	0	0	0	0	0	0	0	0	0 %
Lydian	1	1	1	1	1	1	1	1	100 %
Mixolydian	1	1	0	0	0	1	1	0	50 %
Phrygian	0	0	0	0	0	0	0	0	0 %
Altered Dominant	1	0	0	0	0	1	1	0	38 %
Dorian b2	1	1	1	1	1	1	1	1	100 %
Locrian Natural 9	1	1	1	1	1	1	1	1	100 %
Lydian Aug	1	1	1	1	1	1	1	1	100 %
Melodic Minor	0	0	0	0	0	0	0	0	0 %
Mixolydian #11	1	1	1	1	1	1	1	1	100 %
Mixolydian b6	0	0	0	0	0	0	0	0	0 %
Altered Dominant bb7	1	1	1	1	1	1	1	1	100 %
Dorian #11	1	1	0	0	0	1	1	0	50 %
Harmonic Minor	1	1	1	1	1	1	1	1	100 %
Ionian Aug	1	1	1	1	0	1	1	1	88 %
Locrian Natural 6	1	1	1	1	1	1	1	1	100 %
Lydian #9	1	1	1	1	1	1	1	1	100 %
Phrygian Major	1	1	1	1	1	1	1	1	100 %
Scales Identified Correctly	16	16	14	14	13	17	17	14	
Total of Scales	21	21	21	21	21	21	21	21	
Success Rate by Test	76 %	76 %	67 %	67 %	62 %	81 %	81 %	67 %	

Table 11: Player 4

	Test 1	Test 2	Test 3	Test 4	Test 5	Test 6	Test 7	Test 8	1=Success
	fi= 1.0	fi= 1.1	fi= 1.15	fi= 1.2	fi= 1.25	fi= 1.1	fi= 1.1	fi= 1.1	0=Fail
Player 5	fr= 1.0	fr= 1.2	fr= 1.2	fr= 1.2	fr= 1.2	fr= 1.1	fr= 1.15	fr= 1.25	Success Rate by Scale
Aeolian	0	0	0	0	0	0	0	0	0 %
Dorian	1	1	1	1	1	1	1	1	100 %
Ionian	1	1	1	1	1	1	1	1	100 %
Locrian	1	1	1	1	1	1	1	1	100 %
Lydian	0	1	1	1	1	1	1	1	88 %
Mixolydian	1	1	1	1	1	1	1	1	100 %
Phrygian	0	1	1	1	1	1	1	1	88 %
Altered Dominant	1	1	1	1	1	1	1	1	100 %
Dorian b2	1	1	1	1	1	1	1	1	100 %
Locrian Natural 9	1	1	1	1	1	1	1	1	100 %
Lydian Aug	1	1	1	1	1	1	1	1	100 %
Melodic Minor	1	1	1	1	1	1	1	1	100 %
Mixolydian #11	1	1	1	1	1	1	1	1	100 %
Mixolydian b6	1	1	1	1	1	1	1	1	100 %
Altered Dominant bb7	0	0	0	0	0	0	0	0	0 %
Dorian #11	1	1	1	1	1	1	1	1	100 %
Harmonic Minor	1	1	1	1	1	1	1	1	100 %
Ionian Aug	1	1	1	1	1	1	1	1	100 %
Locrian Natural 6	1	1	1	1	0	1	1	1	88 %
Lydian #9	1	0	0	0	0	0	0	0	13 %
Phrygian Major	1	1	1	1	1	1	1	1	100 %
SUM	17	18	18	18	17	18	18	18	
out of	21	21	21	21	21	21	21	21	
Success Rate by Test	81 %	86 %	86 %	86 %	81 %	86 %	86 %	86 %	

Table 12: Player 5

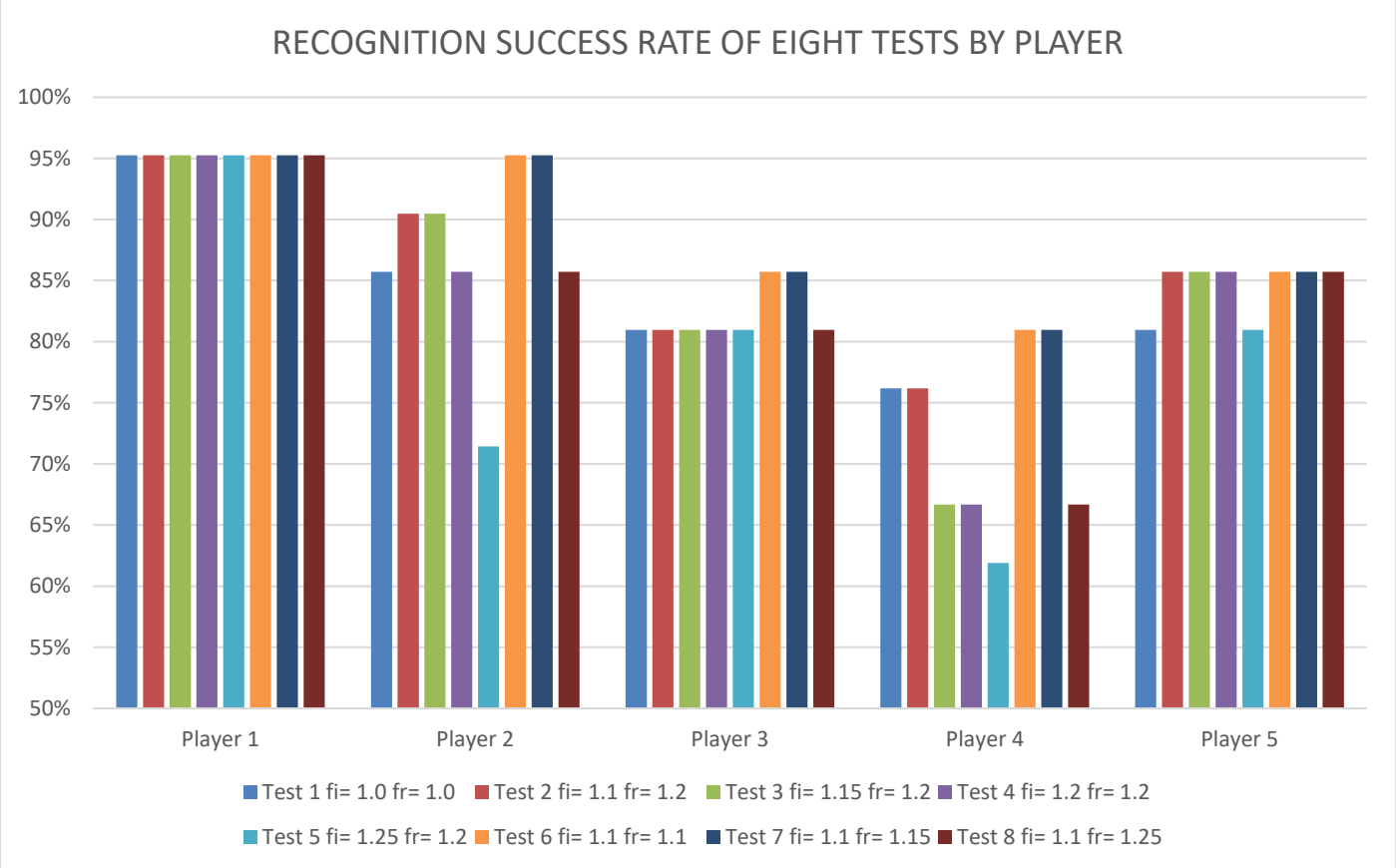


Figure 18: A visual summary of Tables 8-12

4.3 Playing with the system

A way to evaluate the system can also be by playing with it and develop musical approaches to advance its potential. This kind of evaluation comes from personal reflections after I was able to experiment with the system. I would also recommend the reader to see the attached demonstration video and listen to the musical examples in it.

By playing with the system in my rehearsal room, I have noticed that sound is bleeding back from the speakers into the microphone, creating false detection of notes and, eventually, scale outcome. My solution was to use two microphones, one, a condenser mic with low gain for pitch tracking, and the second, a dynamic mic for the acoustic sound of the saxophone. This solution significantly reduced the false detection of pitch and scales. I was standing directly in front of the speakers, which also contributed to the bleeding; therefore, for live performance, I would recommend standing behind the speakers and using headphones for monitoring. In any case, it is essential to be able to clearly hear the auditory output of the system to enhance creativity.

Just like with any musical instrument, device, or controller, mastering requires mastery. Getting used to the controller requires time and practice, but after several hours of playing, my fingers were able to find the buttons instantaneously and without much searching and looking. Another concept that was a bit more difficult to grasp and one that will require more rehearsing is the concept of controlling the harmony by just playing the saxophone. This affordance is new to me, and it is one I have never experienced before. When talking about musical expression with new human-computer interfaces, Dobrian and Koppelman, (2006) state that to reach a level of sophistication achieved by major artists in other specialties like jazz or classical music, it is necessary to encourage further “dedicated participation by virtuosi” to utilize existing virtuosity and develop new virtuosity. A skillful musician with great technical skill and harmonic knowledge, and one that holds an advanced level of virtuosity, will undoubtedly be able to play with the system and benefit from this new affordance of controlling harmonic textures by playing melodic phrases.

From an artistic standpoint, it is exciting and inspiring to play with the system. The long chordal Drone sounds, in combination with the rhythmic notes played by the arpeggiator, provide a subtle and comforting harmonic environment that allows for relaxed and intuitive improvisation. The system is responsive enough that I was able to communicate a scale with both short and long phrases. I was able to identify both correct and false scale matches of the system by looking at the UI on the computer screen and by hearing the sonic output of the system. This situation of staring at the computer screen is not ideal and is addressed in the future work section. While improvising, there were several instances in which the system recognized a different scale than what I communicated or intended to. This can happen due to noise, sound bleeding, or simply wrong playing on my part. I view those instances as a musical challenge and an opportunity to change the harmonic direction of the piece, just as if I would when playing with a live musician. I experimented playing with the Drone and Arpeggiator, both together and separately, and was able

to achieve diverse musical textures that provoked different kinds of playing on my side. Some performance strategies that I have developed by playing with the system over this short period are:

1. Less is more – build the performance by adding and interacting with each sound generating module at a time.
2. Keep in mind that it is possible to start a piece by playing the saxophone first and letting the sound modules join after, or by letting the sound modules play first and joining them after.
3. It is sometimes hard to remember that the system can detect a scale with only a few notes. There is no need to play a lot to convey a scale, and certainly no need to play all the notes in scale order.
4. By setting different speeds, the arpeggiator module provides three kinds of textural material: temporal, melodic, and harmonic. It is easier to play and interact with slower tempos, while rapid tempos can be perceived as harmony to the human ear. Experiment and employ all possibilities.
5. In the current implementation, the system recognizes the scale of a phrase in reference to the first note of the phrase (the bass note). This forces the player to always start the phrase with the bass note, and it is a condition that can be somewhat limiting. A way to evade this is by starting to play a phrase from any note the player wishes to, and only press the capture button before playing the intended bass note of the scale. In this way, the listener will hear a phrase starting on one note, but the system will start capturing the phrase from a different note.

4.4 Future work

The system has great potential to become even more interactive and expressive, and many ideas for improvements and new features come to mind. It is certainly possible to use the system with other monophonic instruments with few minor adjustments to the pitch-detection parameters (windowing, minimum pitch, and downsampling). The Nintendo controller can be replaced by mapping the functions to other sensors or devices like foot pedals. Another approach for communicating the recognized scale by the system, or any other indications the player might need for that matter (capture on/off, sound-generating modules activity, bass note), is to design a small wireless system that will include a battery, a small LCD, and a few LEDs. Having this unit mounted on the saxophone's neck, for example, will allow for the removal of the laptop from the stage, and will offer feedback in a more discrete way.

The system should be further tested and evaluated with a more extensive database of phrases, with as many skillful musicians with a larger variety of instruments. Unsuccessfully detected phrases should be transcribed and analyzed to understand missed matches by the algorithm better. Additional scales should be added into the system for the algorithm to detect from, starting with the seven modes of the Harmonic Major and later with other common miscellaneous scales, including five, six, and eight-note scales.

With minor design modifications, the system can also function as an educational tool and serve aspiring improvisers. Musicians who begin their endeavor of learning jazz improvisation could use this system as a practicing tool by playing a phrase in a specific mode and get a confirmation if the phrase is indeed based on that mode. A substantial part of the education of a jazz musician is ear training or the ability to identify pitch, intervals, melody, chords, rhythm, and other basic elements of music, solely by hearing. For beginners with an undeveloped musical ear, the system can help validate whether they indeed have played a phrase in a particular scale.

From the implementation point of view, the output from the PSM algorithm comes from the JavaScript object (*js scale.js*). The output of the detected scale name is connected to a coll object storing 21 scales by name and pitch-class, and the bass note (or key) of the phrase is being output as a midi integer. This output can be directed into any applications that accept and manipulate a set of notes for note generation and midi effects like arpeggiators, chord, and chord progression makers, melody randomizers, algorithmic composition, and improvisation agents, scale forcing, pitch correction, tuners, and vocoders. Those applications can then be connected to any sound synthesis modules like virtual instruments (VST) or hardware devices like sequencers and synthesizers. The output from the Drone and Arpeggiator modules is a multi-channel audio signal that can be directed into any applications that accept and manipulate audio. The auditory outcome of the system can be tremendously improved by using external effects like reverb, delay, filters, panning, and EQ.

Regarding improvements of existing features, the system should be redesigned to be released from the paradigm that the first note of the phrase represents the key. For that to happen, both the system and the algorithm will require conceptual redesign and extensive work. A musician playing with the system will benefit greatly from having this issue resolved in the future. A better mapping strategy should be developed and implemented to combine existing and future features.

Since the pitch detection module of the system is robust and reliable, indirect gestural acquisition based on continuous tracking of the pitch and amplitude envelop can enhance the system with new features. We can map playing parameters like playing notes in a different register to digital sound processing features like delay, for example. Playing features like high or low, loud or soft, fast or slow can be mapped to control some of the real-valued parameters of the drone and the arpeggiator to affect the system's auditory outcome. It is also possible to add direct gestural acquisition functionality by adding sensors and implementing a more sophisticated way to control input information, for example, one-to-many mapping, can take the system to a new stage of instrument augmentation.

The implementation of the system is available for users and developers as free and open-source software.[26] Appendix D contains all necessary information regarding the available software, including system requirements, a list of related files, and an operation manual.

[26] PSM System repository on github - https://github.com/guysion/PSM-Interactive-Music-System

5. Conclusion

The interest of conveying a harmonic environment to a computer by improvising a melody has led me to develop a system that augments the saxophone in a new and intriguing way. By reviewing existing saxophone augmentation works, I recognized the absence of an approach that grants monophonic instrumentalists with the ability to control the harmonic outcome of an interactive music system for music-making. Previous augmentation-works almost exclusively concentrated on enhancing the saxophone with various sensors designed to directly control sound synthesis, digital signal processing, effects, or even multimedia. Coming from the world of jazz and improvised music, where harmonic ideas are communicated among players instantaneously and in a nonverbal way while playing together, I focused on awarding a machine with the musical skill of recognizing a scale from a melody.

Motivated by the work of Rowe (2004) in 'Machine musicianship', I recognized the necessity for an algorithm that would perform real-time low-level analysis of musical input to extract a scale from an improvised phrase. The Phrase to Scale Match (PSM) algorithm presented in this thesis can analyze a monophonic improvised phrase rooted in a single harmonic environment without modulation. The PSM algorithm may be regarded as this thesis most significant contribution to the field of music algorithmic analysis. When discussing the "paradox at the heart of the transfer of musical knowledge to a machine," Rowe (2004) points out the extensive work required to "make a computer program perform the analysis required of a freshman music student," while with music systems, "a little musical knowledge goes a long way." Rowe's measure of success regarding music programs, and certainly a measure I endorse, "is not whether these programs match empirical data from research with human subjects, but whether they output structures that make musical sense." The PSM algorithm resembles an ear-trained musician with functional pitch recognition, a skill which involves identifying the relationship between a pitch and an established harmonic context. The algorithm has been evaluated with an audio database consisting of 105 improvised musical phrases played by five saxophonists, with a successful matching rate of 89%. While certainly having some room for improvement, the level of accuracy seemed sufficient enough to try to evaluate the system as a whole by playing with it.

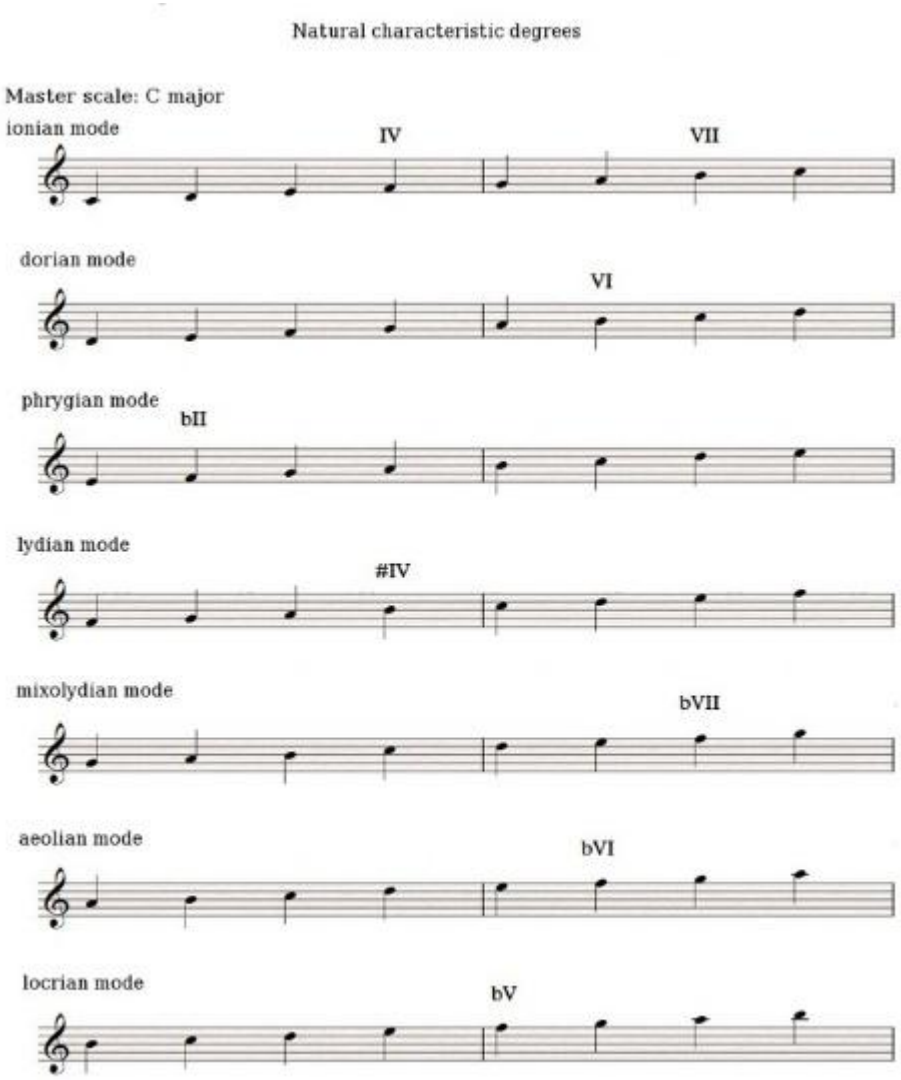
The interactive music system integrating the PSM algorithm also presents third-party components taken off-the-shelf and customized for this specific application, such as the pitch tracking module processing the audio input. The system enables the musician to determine the length of the phrase to be analyzed by pressing the capture button via a dedicated tangible controller mounted on the saxophone. The captured phrase is processed and analyzed by the PSM algorithm, and the result, in the form of a scale name, is applied to be used for playing music in an interactive way. Following Rowe's (2004) premise that "adding an auditory component to interactive systems brings them much closer to the human experience of music," two sound generating modules were developed. A drone and arpeggiator sound modules were added to the system to explore and present a practical way to play and interact with the system, and to expose the reader/listener to the interactive

possibilities the system can offer. The result of this explorative interaction can be seen and heard in a demo video.[27]

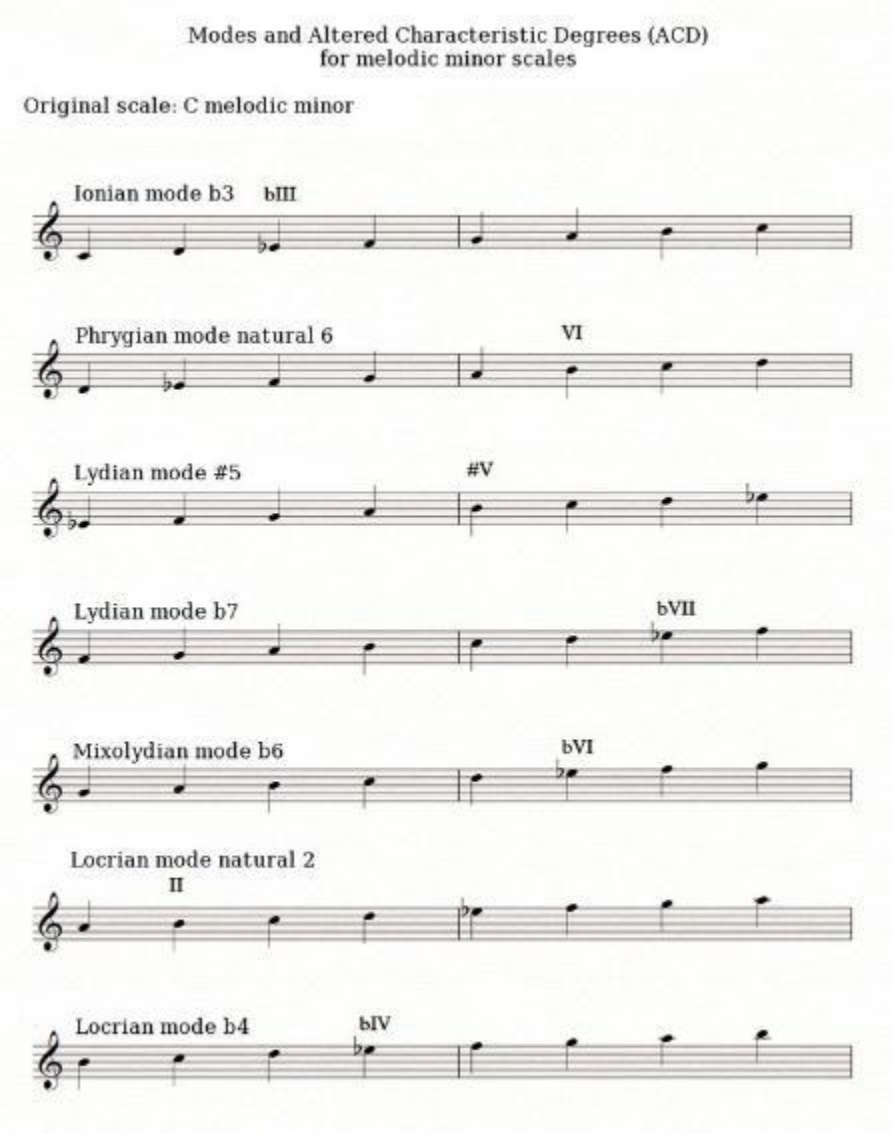
To conclude, the work presented in this thesis successfully addresses my original research question. The algorithm is able to recognize the scale of a musical phrase with high accuracy, and the system offers the musician the ability to influence the harmonic output by merely playing an improvised phrase. The degree of randomization applied to the sound generating modules creates the impression of several entities collectively generating music and sharing the creative control, in almost the same manner as a group of live musicians would do. The outcome of this thesis provides musicians playing monophonic instruments a novel way to communicate harmony to a machine, in such a way that other algorithms and applications can use this information to contribute to the musician-driven sonic creative process.

[27] Demo video of the system - https://youtu.be/u-ObVjojjyc

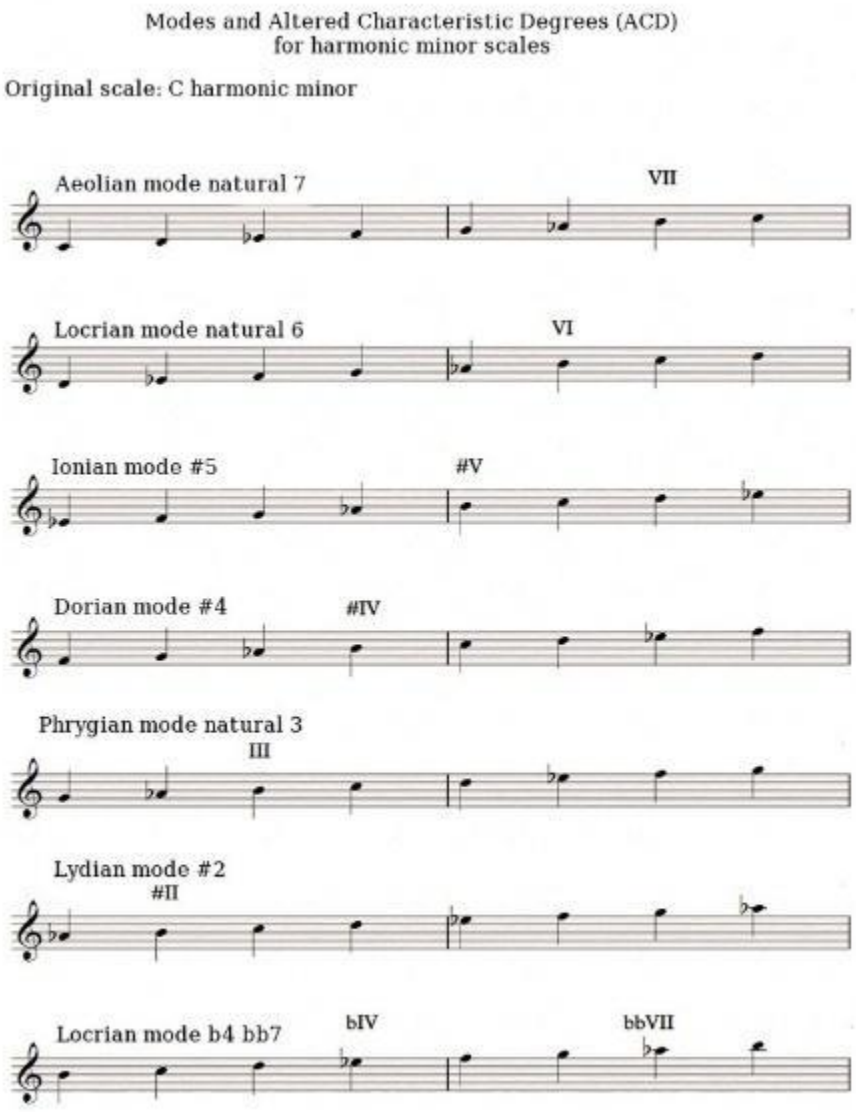
Appendix A



Appendix A-1. The natural modes of the major scale and their natural characteristic degrees



Appendix A-2. Altered modes from the melodic minor scale distinguish and their altered characteristic degrees



Appendix A-3. Altered modes from the harmonic minor scale and their altered characteristic degrees

Appendix B

Dear Saxophonist,

For my finale project, I developed a system and an algorithm for estimating the mode/scale of an improvised musical phrase. The main idea is for us saxophone players to control a system that can, for example, accompany us, based on our musical output. The way the system is built is that it tracks the pitch, converts it to midi, analyzes, and processes the notes in the phrase and output an estimated scale, which can then be used for a musical drone model, arpeggiator, or chord progression. I am looking for as many Acappella audio clips of skillful saxophone players improvising on different keys and scales. If you have the time to help me, here are some comments and guidelines to make my job easier and your contribution meaningful:

1. Try to play the phrase like you are trying to convey the key and mode to a band that is with you on stage.

2. The first note of the phrase must be the first degree of the scale you are playing in. So, for example, if you are playing a phrase in D Dorian, the first note of the phrase *must* be D.

3. The phrase must be based on only one key and scale mode.

4. It is better if you can avoid super-fast phrases because it would make it hard on the system to track the pitch accurately. Also, for the same reason, try to avoid bending notes.

5. A phrase can be long or short, can contain chromaticism, and may or may not have all the notes of the scale at hand. My best suggestion is to keep in mind the 'playing with a band' and trying to convey the key and mode or playing a cadenza in one key and mode.

6. You can record it on your phone, but better sound quality would be my preferred choice, so try to record it with a mic or a zoom recorder with a good volume level and without reverb.

7. Make sure that you label the sound files correctly with the key and mode info, for example, F Mixolydian #11 or C Dorian b2.

8. I am attaching to this email a pdf with the 21 scales that will form my database. Seven modes times 3 (Major, Melodic Minor, Harmonic Minor), so please have a look at it and used it if you need to. You do not need to record all modes, but I would appreciate it if you could.

9. The audio clips will not be used for anything else besides evaluating the algorithm.

10. A zip file uploaded to Dropbox or google drive containing three folders (Major, Minor melodic, Minor harmonic) with at least seven improvised phrases (1 for each mode, if not more) would be the best.

I should also mention that in the whole process, no personal data will either be stored or disclosed to the public. Data will be collected and stored anonymously in compliance with GDRP.
In case this was not clear enough, please feel free to drop me a line. I appreciate your help!

Best regards,

Guy Sion

Appendix C

Chord Scales

Major Scale Modes

Scale Degree	Mode	Relation to Major Scale							Most Common Use
I, I^{maj7}	Ionian	1	2	3	4	5	6	7	Maj^{7}
ii, ii^{7}	Dorian	1	2	b3	4	5	6	b7	$Min^{7(nat.6)}$
iii, iii^{7}	Phrygian	1	b2	b3	4	5	b6	b7	Min^{7}, MajΔ/7
IV, IV^{maj7}	Lydian	1	2	3	#4	5	6	7	$Maj^{7(\#11)}$
V, V^{7}	Mixolydian	1	2	3	4	5	6	b7	Dom^{7}
vi, vi^{7}	Aeolian	1	2	b3	4	5	b6	b7	$Min^{7(b6)}$
vii°, vii^{7b5}	Locrian	1	b2	b3	4	b5	b6	b7	Min^{7b5}

Melodic Minor Modes

Scale Degree	Mode	Relation to Major Scale							Most Common Use
i, i^{maj7}	Melodic Minor	1	2	b3	4	5	6	7	Min^{maj7}
ii, ii^{7}	Dorian b2	1	b2	b3	4	5	6	b7	$Min^{7\,sus4b9}$
bIII+, $bIII^{+maj7}$	Lydian Aug.	1	2	3	#4	#5	6	7	$Maj^{7\#4\#5}$, MajΔ/b6
IV, IV^{7}	Mixolydian #11	1	2	3	#4	5	6	b7	Dom^{7b5}
V, V^{7}	Mixolydian b6	1	2	3	4	5	b6	b7	Dom^{7b6}
vi°, vi^{7b5}	Locrian Nat.9	1	2	b3	4	b5	b6	b7	Min^{9b6}
vii°, vii^{7b5}	Altered Dominant	1	b2	b3	b4	b5	b6	b7	$Dom^{7b9,\#9,b5,\#5}$

Harmonic Minor Modes

Scale Degree	Mode	Relation to Major Scale							Most Common Use
i, i^{maj7}	Harmonic Minor	1	2	b3	4	5	b6	7	Min^{maj7}, oΔ/b7
ii°, ii^{7b5}	Locrian Nat.6	1	b2	b3	4	b5	6	b7	Min^{7b5}
bIII+, $bIII^{+maj7}$	Ionian Aug.	1	2	3	4	#5	6	7	$Maj^{7sus4,\#5}$
iv, iv^{7}	Dorian #11	1	2	b3	#4	5	6	b7	$Min^{7(\#11)}$
V, V^{7}	Phrygian Major	1	b2	3	4	5	b6	b7	$Dom^{7sus4,b9,\#5}$
Vi, Vi^{maj7}	Lydian #9	1	#2	3	#4	5	6	7	$Maj^{7\#9,\#11}$, MajΔ/b9
vii°, vii^{o7}	Altered Dominant bb7	1	b2	b3	b4	b5	b6	bb7	Dim^{o7}

Appendix D

Phrase-to-Scale (PSM) Interactive Music System

The PSM Interactive Music System offers musicians who play monophonic instruments the ability to control the harmonic output of the system by tracking their melodic output.

Inquiries: guysionmusic@gmail.com

REQUIREMENTS
The PSM Interactive Music System is implemented in Cycling '74 Max 8 (Windows 10 64-bit).
The PSM Interactive Music System has been developed and tested on the following platform:
Windows 10 (64-bit)
MAX 8.1.4 (64-bit)

The following Cycling '74 Max 8 libraries are required: (download the externals and include these in your Max path)
OMax.Yin+core.maxpat (to be placed in patcher subfolder)
bc.yinstats.mxo (to be placed in externals subfolder)
Download from:
OMax https://forum.ircam.fr/projects/detail/omax/
or from https://github.com/DYCI2/OMax4

yin~.mxe64 or yin~.mxe (to be placed in externals subfolder)
Download from:
Max Sound Box https://forum.ircam.fr/projects/detail/max-sound-box/

INCLUDED FILES
PSM System.maxpat - main Max patch
scale.js - PSM algorithm code in JavaScript
scale.txt - Max coll file containing scale names and pitch class of each scale
Nintendo black white.jpg - Nintendo controller illustration
PSM System.maxproj - Max project file
Aeolian. G.wav, Dorian. G.wav, Mixolydian. G.wav - 3 audio examples of phrases in different scales
NOT INCLUDED FILES
OMax.Yin+core.maxpat
bc.yinstats.mxe64
yin~.mxe64

MANUAL
Demo video of the system is available here: https://youtu.be/u-ObVjojjyc
Run the PSM_System.maxpat

Audio I/O Module: Press the speaker icon (ezdac~ object) to turn the audio on.
Choose the type of input from the scroll down menu.
Live Mode to be used for live performance.
Sample Mode to be used for playing audio samples from the playlist module.
Turn Monitor on if you wish to hear yourself in the mix, the note tracker will work regardless.
Adjust microphone input level by using the visual level indicator and the gain dial.

Pitch detection Module: Play a few notes and check if your pitch is being tracked, notes should appear on the staff line and the velocity meter should be moving.
Noise Thresh - Defines a quality factor under which the input signal is rejected from the pitch detection, similar to a noise threshold.
Consistency - Directly related to the estimation quality factor coming from the yin~, meaning, pitches with estimated quality under the value given, will be ignored.
Window (ms) - Defines a time window after which an onset is validated if the estimated pitch remains stable during that time.
Min pitch - set the lowest note of your instrument: Soprano sax-G#2, Alto sax- C#2, Tenor sax-G#1, Baritone sax with low Bb-C#1.
Down sampling – 0=off, 1/2/3=downsampling by 2/4/8. Keep in mind the tradeoff that high downsampling lowers not only the computation cost but also the reliability of the estimation.

PSM Module: Press the capture button and play a phrase in one mode/scale/harmonic environment. Keep in mind that the first note of the phrase should be the tonic/key of the mode/scale you are trying to establish. Try to avoid bending note.
Press the capture button again. The recognized scale and the bass note should appear. You can view the notes that are being captured in the dedicated staff line.

Drone Module: Once a scale has been recognized by the system, press the Drone button to turn it on. Adjust the level with the Gain dial.

Arpeggiator Module: Once a scale has been recognized by the system, press the Arp button to turn it on. Adjust the level with the Gain dial. You can also adjust the speed of the arpeggiator with the Arp Speed Dial.

Playlist: To analyze audio samples of improvised musical phrases, drag the audio files into the playlist object and press play. Keep in mind that the input mode should be set Sample Mode.

Controller: It is possible to use the system with any controller. At the moment the mapping is set as such:
Capture on/off – yellow key**,** Drone on/off – green key**,** Arpeggiator on/off – red key
Increase Arp speed – D-Pad up key**,** Decrease Arp speed – D-Pad down key
Increase Arp speed step size – D-Pad left key, Decrease Arp speed step size – D-Pad right key

References

Andreas, M. (1988) ‘Softwind Synthophone (MT Nov 1988)’, *Music Technology*. Music Maker Publications (UK), Future Publishing., (Nov 1988), pp. 74–76.

Babacan, O. *et al.* (2013) ‘A comparative study of pitch extraction algorithms on a large variety of singing sounds’, in *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*. IEEE, pp. 7815–7819.

Burtner, M. (2002) ‘Noisegate 67 for metasaxophone: composition and performance considerations of a new computer music controller’, in *Proceedings of the 2002 conference on New interfaces for musical expression*. Citeseer, pp. 1–6.

Casey, M. A. *et al.* (2008) ‘Content-based music information retrieval: Current directions and future challenges’, *Proceedings of the IEEE*, 96(4), pp. 668–696.

Cook, P. R. (1992) ‘An Automatic Pitch Detection and MIDI Control System for Brass’.

Cook, P. R. (2009) ‘Re-Designing Principles for Computer Music Controllers: a Case Study of SqueezeVox Maggie.’, in *NIME*, pp. 218–221.

De Cheveigné, A. and Kawahara, H. (2002) ‘YIN, a fundamental frequency estimator for speech and music’, *The Journal of the Acoustical Society of America*, 111(4), pp. 1917–1930.

De La Cuadra, P., Master, A. S. and Sapp, C. (2001) ‘Efficient Pitch Detection Techniques for Interactive Music.’, in *ICMC*.

Dobrian, C. and Koppelman, D. (2006) ‘The’E’in NIME: Musical Expression with New Computer Interfaces.’, in *NIME*, pp. 277–282.

Favila, S. (2008) ‘Gluisax: Bent Leather Band’s Augmented Saxophone Project’, p. 4.

Flores, C. R., Murphy, J. and Norris, M. (2019) ‘HypeSax: Saxophone acoustic augmentation’.

Gao, Q. (2015) ‘PITCH DETECTION BASED MONOPHONIC PIANO TRANSCRIPTION’, *Yankee*, 7(60), p. C4.

Gerhard, D. (2003) *Pitch extraction and fundamental frequency: History and current techniques*. Department of Computer Science, University of Regina Regina, Canada.

Ghahremani, P. *et al.* (2014) ‘A pitch extraction algorithm tuned for automatic speech recognition’, in *2014 IEEE international conference on acoustics, speech and signal processing (ICASSP)*. IEEE, pp. 2494–2498.

Hong, E., Kim, B. and Han, K. (2016) ‘Dr. Saxophone: Hybrid saxophone interface’, in *2016 3rd International Conference on Systems and Informatics (ICSAI)*. IEEE, pp. 1149–1153.

Hong, E. and Kim, J. (2017) ‘Telesaxophone: Hybrid saxophone Interface’, in *Proceedings of the International Conference on Algorithms, Computing and Systems*, pp. 83–87.

Kennan, K. W. (1972) *COUNTERPOINT: BASED ON EIGHTEENTH-CENTURY PRACTICE/KENT WEELER KENNAN*.

von dem Knesebeck, A. and Zölzer, U. (2010) ‘Comparison of pitch trackers for real-time guitar effects’, in *Proc. 13th Int. Conf. Digital Audio Effects*.

Lévy, B. (2004) ‘OMax’. Available at: https://github.com/DYCI2/OMax4/blob/master/OMax4-Doc.pdf.

Longuet-Higgins, H. C. and Steedman, M. J. (1987) ‘On interpreting Bach’, *Mental processes: Studies in cognitive science*. British Psychological Society/MIT Press London and Cambridge, MA, pp. 82–104.

Lyons, R. G. (2004) *Understanding digital signal processing, 3/E*. Pearson Education India.

Manzo, V. J. (2007) ‘Implementing Modality in Algorithmic Composition’.

Manzo, V. J. (2016) *Max/MSP/Jitter for music: A practical guide to developing interactive music systems for education and more*. Oxford University Press.

Melo, J., Gómez, D. and Vargas, M. (2012) ‘Gest-O: Performer gestures used to expand the sounds of the saxophone.’, in *NIME*.

Robertson, A. (2014) ‘Bassline Pitch prediction for real-time performance systems’, in *ICMC*.

Rowe, R. (2004) *Machine musicianship*. MIT press.

Schiesser, S. and Traube, C. (2006) ‘On making and playing an electronically-augmented saxophone’, in *Proceedings of the 2006 conference on New interfaces for musical expression*, pp. 308–313.

Softwind Instruments (1986) *The Synthophone Manual*.

Suk, S.-Y., Chung, H.-Y. and Kojima, H. (2007) ‘Voice/non-voice classification using reliable fundamental frequency estimator for voice activated powered wheelchair control’, in *International Conference on Embedded Software and Systems*. Springer, pp. 347–357.

Temperley, D. (1999) ‘What’s key for key? The Krumhansl-Schmuckler key-finding algorithm reconsidered’, *Music Perception*. University of California Press, 17(1), pp. 65–100.

Traube, C., Depalle, P. and Wanderley, M. (2003) ‘Indirect acquisition of instrumental gesture based on signal, physical and perceptual information’, in *Proceedings of the 2003 conference on New interfaces for musical expression*, pp. 42–47.

Vasilik, M., Stillings, L. and Cortazar, C. (2015) ‘Pitch Detection for Music in Noisy Environments’.

Zahorian, S. A. and Hu, H. (2008) ‘A spectral/temporal method for robust fundamental frequency tracking’, *The Journal of the Acoustical Society of America*. Acoustical Society of America, 123(6), pp. 4559–4571.

Zhao, L. (2016) ‘Musical Scales Recognition via Deterministic Walk in a Graph’, in *2016 5th Brazilian Conference on Intelligent Systems (BRACIS)*. IEEE, pp. 151–156.