Friedrich-Alexander-Universität
Faculty of Engineering

Dissertation

2025

Peter Meier

Real-Time Algorithms for Beat Tracking and Pitch Estimation

Real-Time Algorithms for Beat Tracking and Pitch Estimation

Echtzeit-Algorithmen zur Beat-Erkennung und Tonhöhen-Schätzung

Dissertation

Der Technischen Fakultät
der Friedrich-Alexander-Universität
Erlangen-Nürnberg
zur
Erlangung des Doktorgrades Dr.-Ing.
vorgelegt von

Peter Meier
aus Regensburg

Als Dissertation genehmigt
von der Technischen Fakultät
der Friedrich-Alexander-Universität Erlangen-Nürnberg

Tag der mündlichen Prüfung: 3. Dezember 2025

1. Gutachter:	Prof. Dr. rer. nat. Meinard Müller
2. Gutachter:	Prof. Dr.-Ing. Christof Weiß

Declaration about the usage of generative AI and AI-assisted technologies

During the preparation of this dissertation, the author used ChatGPT[1] in order to check grammar, ensure style consistency, assist with rephrasing, and improve readability. After using this tool, the manuscript was carefully reviewed and the content was edited as needed. No tools or services were used for content generation. The content of this dissertation was entirely created by the author, who has ensured that all material presented reflects their original work and research. The author takes full responsibility for the content of the dissertation.

[1] https://chatgpt.com

Abstract

Music Information Retrieval (MIR) is an interdisciplinary field focusing on computational methods for analyzing and interacting with music. A key application area in MIR is the development of interactive music systems that respond to musical input in real time, enabling new forms of expression in live performance, music education, and music-based gaming. These systems rely on real-time MIR techniques that must be causal, efficient, and robust, operating within strict timing constraints, on limited hardware, and under real-world conditions such as background noise and instrument cross-talk. This thesis focuses on two fundamental musical elements, rhythm and melody, and proposes real-time algorithms for beat tracking and pitch estimation tailored for interactive use. The first part of this thesis focuses on real-time beat tracking, which involves estimating the temporal structure of music by identifying beat positions, which is a fundamental component for synchronization and interaction in live musical settings. Our key contribution is the adaptation of predominant local pulse (PLP) estimation from offline analysis to real-time use. The proposed algorithm operates with zero latency, continuously estimates beat positions, and provides stability metrics along with lookahead features for predicting upcoming beats. These enhancements enable the generation of pulse-based control signals that drive beat-synchronous audio effects, supporting dynamic interaction in live performance and mixing. The system is demonstrated in various practical contexts, including interactive music-making tools, rhythm-based educational games, and creative production environments. The second part of this thesis addresses real-time pitch estimation, focusing on tracking the fundamental frequency of musical signals in interactive settings. Our main contribution is `RT-SWIPE`, a real-time adaptation of the established `SWIPE` algorithm. This version is optimized for low-latency, multi-channel use and balances accuracy with computational efficiency to support robust pitch tracking in ensemble scenarios. To evaluate performance under real-time constraints, a delay-tolerant metric is introduced. This metric extends the standard raw pitch accuracy (RPA) by accounting for algorithmic delays. The system is further examined in the context of pitch estimation for wind instruments, where cross-talk presents a common challenge. Based on these insights, a real-time intonation monitoring system is developed for wind ensembles. It provides immediate, individualized pitch feedback to musicians, supporting ensemble tuning and live musical communication. The system is tested in a real-world rehearsal environment, demonstrating its practical value. Finally, a music-based game prototype is presented, using real-time pitch input to enable interactive, creative gameplay. This showcases the broader potential of the system in educational and entertainment contexts. Through systematic evaluation, real-world testing, and application prototypes, this thesis advances real-time MIR with lightweight, responsive, and robust algorithms and systems that enhance musical performance, learning, and creative expression.

Zusammenfassung

Music Information Retrieval (MIR) ist ein interdisziplinäres Forschungsfeld, das sich mit rechnergestützten Methoden zur Analyse von Musik und zur Interaktion mit Musik beschäftigt. Ein zentrales Anwendungsgebiet ist die Entwicklung interaktiver Musiksysteme, die in Echtzeit auf musikalische Signale reagieren und neue Ausdrucksformen für Live-Auftritte, Musikpädagogik und musikbasierte Spiele ermöglichen. Solche Systeme erfordern kausale, effiziente und robuste Verfahren, die unter strengen Zeitvorgaben, auf limitierter Hardware und auch unter realen Bedingungen wie Hintergrundgeräuschen zuverlässig funktionieren. Diese Dissertation konzentriert sich auf zwei grundlegende musikalische Aspekte, Rhythmus und Melodie, und stellt Echtzeit-Algorithmen zur Beat-Erkennung und Tonhöhen-Schätzung vor, speziell für interaktive Anwendungen. Im ersten Teil widmen wir uns der Echtzeit-Beat-Erkennung, die die zeitliche Struktur von Musik erfasst, indem Beat-Positionen identifiziert werden, was entscheidend für Synchronisation und Interaktion in Live-Situationen ist. Unser Beitrag besteht in der Weiterentwicklung der Predominant Local Pulse (PLP) Methode für Echtzeitanwendungen. Das ursprünglich für Offline-Analysen entwickelte Verfahren wurde so angepasst, dass es latenzfrei arbeitet, kontinuierlich Beat-Positionen schätzt und Stabilitätsmetriken sowie Vorhersagen künftiger Beats liefert. Darauf aufbauend erzeugen wir pulsbasierte Steuersignale für beat-synchrone Audioeffekte, die dynamische Interaktionen bei Live-Mixing und Auftritten ermöglichen. Die Methoden werden in praxisnahen Szenarien demonstriert, unter anderem durch Werkzeuge für interaktives Musizieren, Musikspiele und kreative Produktionsumgebungen. Im zweiten Teil dieser Dissertation widmen wir uns der Echtzeit-Tonhöhen-Schätzung, die die Grundfrequenz musikalischer Signale in Echtzeit ermittelt. Wir präsentieren `RT-SWIPE`, eine Echtzeitversion des etablierten `SWIPE`-Algorithmus, optimiert für latenzarme Mehrkanalanwendungen. `RT-SWIPE` vereint Genauigkeit mit Effizienz, um robuste Tonhöhenerkennung in Ensembleszenarien zu ermöglichen. Zur Evaluation unter Echtzeitbedingungen wird eine Metrik eingeführt, die Verzögerungen berücksichtigt und die Raw Pitch Accuracy (RPA) entsprechend erweitert. Zudem untersuchen wir die Auswirkungen von Übersprechen bei Blasinstrumenten und entwickeln ein Echtzeitsystem, das Musizierenden unmittelbares, individuelles Feedback zur Intonation bietet. Die Praxistauglichkeit des Systems wird in einer realen Probesituation evaluiert und zeigt Potenzial zur Verbesserung der Intonation und musikalischen Kommunikation im Ensemble. Abschließend entwickeln wir ein musikbasiertes Spiel, das Echtzeit-Tonhöheneingaben für Interaktivität und Kreativität nutzt, was die Anwendbarkeit des Verfahrens in Bildung und Unterhaltung verdeutlicht. Durch systematische Evaluierung, Praxistests und Anwendungsprototypen trägt diese Dissertation zur Weiterentwicklung der Echtzeit-MIR bei; mit effizienten, reaktionsschnellen und robusten Algorithmen und Systemen, die musikalische Darbietung, Lernen und kreativen Ausdruck fördern.

Contents

1 Introduction

Music has a unique place in human culture. It transcends language and cultural boundaries, acting as a universal medium of expression. Through rhythm, melody, and harmony, people have long used music to share emotions, tell stories, communicate ideas, and build connections. Even in our highly digital world, music retains this timeless ability to move and unite us.

As music increasingly exists in digital form, the need to understand and interact with it in new ways has grown. In response to this evolving landscape, **Music Information Retrieval (MIR)** brings together interdisciplinary research from fields such as signal processing, machine learning, informatics, musicology, and psychoacoustics. This enables the development of algorithms for a variety of music-related analysis tasks, such as beat tracking [9, 30, 50, 139], tempo estimation [13, 43, 117], music synchronization [38, 45], pitch estimation [21, 35, 66, 82, 108], chord recognition [3, 72], harmony analysis [48, 71], automatic music transcription [55, 80, 136], music search and recommendation [15, 115], source separation [57, 129], music classification [73, 97], and many more. By leveraging computational methods, MIR enables detailed analysis of musical data and supports new, interactive approaches to engaging with music.

Building on these advances, technology has transformed how we experience music, making **interactive music systems** a growing area of interest. These systems enable real-time interaction across a variety of contexts, including live performance tools that respond dynamically to musicians on stage [4, 102, 131], educational platforms that support and guide students in learning an instrument [16, 36], immersive listening experiences and artistic sound installations [56, 101], music games that engage users through interactive challenges [39, 77], and adaptive systems for real-time sound generation [17, 64]. For these systems to feel natural and responsive, they rely on algorithms that can process music data efficiently and accurately.

Real-time audio processing is therefore a critical requirement in interactive music applications, where systems must respond instantaneously. Unlike offline analysis, real-time algorithms operate under strict constraints [127]: The system must process audio data causally, operating incrementally on each block of incoming samples without access to future data. Each block must be processed efficiently, meeting strict timing constraints and running on limited hardware resources. These conditions significantly complicate algorithmic design and the implementation of usable software that can reliably support musicians in real-time scenarios.[2] Techniques that work well in batch or offline settings, where post-processing and

[2] `http://www.rossbencina.com/code/real-time-audio-programming-101-time-waits-for-nothing`

latency are acceptable, often fail when applied in real-time contexts. Algorithms must be not only causal and efficient, but also robust to real-world conditions such as background noise and instrument cross-talk [127, 138].

Musical signals are often noisy, unpredictable, and highly variable, particularly in environments such as rehearsal rooms, live stages, crowded venues, or interactive sound installations, where ambient noise and cross-talk between instruments can complicate analysis [20, 27, 105]. A real-time system must remain stable and accurate despite these uncertainties, avoiding erratic behavior or breakdowns. Together, these three challenges—**causality**, **efficiency**, and **robustness**—make real-time MIR [127] a particularly demanding but highly rewarding field of research.

At its core, music signals unfold along two fundamental dimensions: time and frequency. The **time domain** encompasses rhythm, tempo, and the sequencing of musical events, while the **frequency domain** captures pitch, melody, and harmony. These two aspects are central to how we perceive and interpret music. As musician Jacob Collier[3] puts it:

> *"Music can be defined by three basic forces. There's **rhythm**, which is basically the body; there's **harmony**, which is like the relationship between things; and then there's **melody**, which is like the voice."*

This framing resonates with both musical intuition and computational analysis. To access these musical "forces" algorithmically in real time, this thesis focuses on two core tasks in MIR: beat tracking and pitch estimation.

Real-time beat tracking analyzes the timing of musical events to detect tempo and rhythmic structure, the "body" of music, as described by Collier, enabling synchronization and interaction [23, 58]. **Real-time pitch estimation** extracts the fundamental frequencies of sound, revealing melodic and harmonic content, the "voice" and "relationships" in music, necessary for transcription, melody tracking, and tonal analysis [6, 35]. Together, these methods are important building blocks for interactive music systems that respond to both rhythm and melody.

The potential applications of such technology are wide-ranging. Beyond interactive music systems, real-time MIR enables new forms of creative expression [67, 70, 121], supports advanced music education tools [36, 54], facilitates automatic accompaniment [31, 33, 44, 68, 137], and powers real-time audio effects [125, 126, 128]. The ability to analyze and respond to music in real time opens up possibilities for both artistic innovation and practical solutions in diverse musical contexts.

Despite notable advances in MIR, challenges remain in adapting many methods to real-time use. Most existing algorithms are developed and evaluated in offline settings, and may not directly satisfy the latency, causality, and robustness constraints of interactive music systems that operate on live audio. As a result,

[3] Jacob Collier, *Little Blue, Djesse Vol. 4, Music Theory*, YouTube video, posted by Zach Sang Show (@zachsangshow), 28 September 2023, timestamp 15:59, `https://www.youtube.com/watch?v=6peXQsJHtOw&t=959s`.

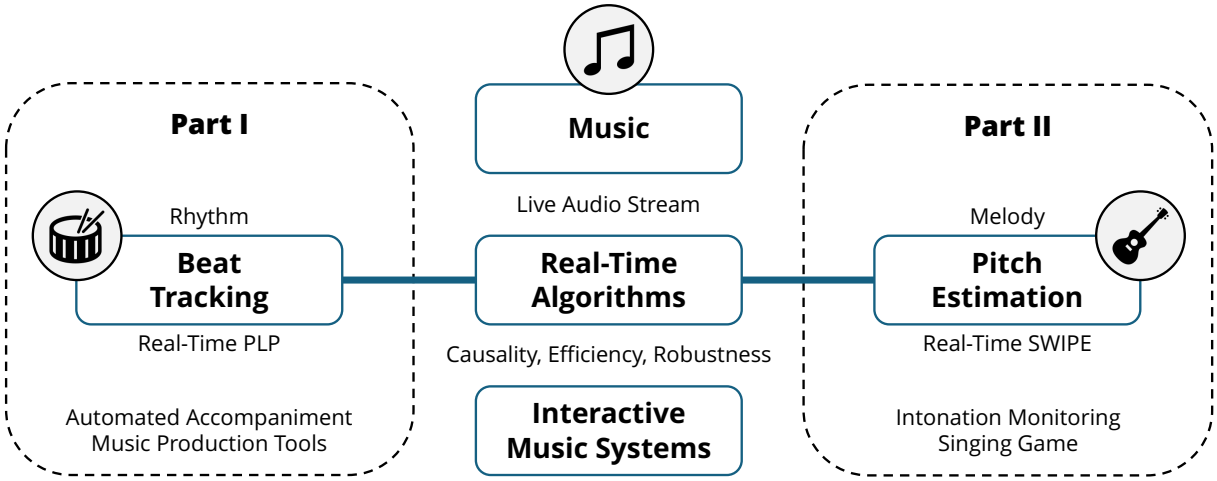


Figure 1.1: Real-time algorithms for beat tracking and pitch estimation enable interactive music systems that respond to live audio, enhancing musical performance, education, and creative expression.

techniques that achieve strong performance in offline evaluations can encounter difficulties meeting the practical demands of real-time, interactive scenarios.

This thesis addresses these challenges by developing real-time algorithms for beat tracking and pitch estimation, tailored specifically for interactive music applications. By focusing on both the rhythmic and melodic dimensions of music, two of its most fundamental elements, the work seeks to contribute a practical basis for improving the responsiveness, reliability, and usability of MIR in interactive contexts. The proposed methods build on and extend existing approaches, adapting them for causal, low-latency operation. Comprehensive evaluations against state-of-the-art techniques are presented, alongside real-world testing and user studies. To support these investigations, a set of interactive application prototypes is developed to demonstrate and assess the algorithms under realistic conditions, and to provide a resource for musicians and developers interested in exploring the interactive use of music signals.

In summary, this thesis presents real-time algorithms for beat tracking and pitch estimation that are specifically designed to meet the challenges of causality, efficiency, and robustness under real-world conditions, enabling reliable and responsive applications in interactive music systems.

1.1 Structure and Main Contributions of this Thesis

The following section provides an overview of the thesis structure and summarizes its main contributions. This thesis is organized into two main parts, as illustrated in Figure 1.1: **Part I** focuses on real-time beat tracking, while **Part II** addresses real-time pitch estimation. Each part presents methods, evaluations, and applications specific to the respective task.

Before considering individual tasks, in **Chapter 2**, we establish the technical and mathematical fundamentals necessary for understanding real-time music information retrieval. We cover the basics of audio signals, the constraints of real-time audio processing, and evaluation metrics for interactive music systems, providing a unified notation for the methods and analyses that follow.

Next, in **Chapter 3**, a novel real-time approach for beat tracking based on the predominant local pulse (PLP) concept is presented. The transformation of the original offline PLP algorithm into a dynamic, low-latency system suitable for interactive music applications is detailed, with technical challenges addressed and new features such as beat lookahead and stability metrics introduced.

Building on the real-time PLP method, **Chapter 4** develops a set of pulse-based control signals for beat-synchronous audio effects. Normalization techniques are introduced to disentangle pulse oscillation from stability, enabling the extraction of low-frequency oscillators and confidence measures directly from the PLP output. These control signals support dynamic and adaptive control of audio effect parameters in real time, with applications in live mixing, performance, and education.

To demonstrate the practical impact of the developed methods, **Chapter 5** showcases interactive applications that leverage real-time beat tracking and pulse-based control signals. Prototypes for music making, educational gaming, and creative music production are presented, highlighting the versatility and responsiveness of the algorithms in both artistic and educational contexts.

Shifting focus to pitch estimation, **Chapter 6** presents a preliminary study where we evaluate lightweight algorithms such as `YIN` and `SWIPE` for their suitability in real-time interactive music applications. Through systematic experiments and comparisons, we highlight the strengths and limitations of each approach, laying the groundwork for further development and creative use.

Further refinement is explored in **Chapter 7**, where we adapt the `SWIPE` algorithm for real-time pitch estimation. We introduce `RT-SWIPE`, a causal variant designed for low-latency, multi-channel processing, and provide a detailed analysis of its computational efficiency and responsiveness, supported by new evaluation metrics and experimental results.

Robustness in real-world scenarios is addressed in **Chapter 8**, where we systematically analyze the challenge of cross-talk in ensemble recordings. Using multi-track wind instrument recordings, we compare the robustness of various pitch estimation algorithms and propose practical solutions to mitigate interference, paving the way for reliable real-time intonation monitoring systems.

Building on the insights from previous chapters, in **Chapter 9** we examine interactive pitch estimation applications. We present real-time systems for intonation monitoring in music ensembles, introduce creative music games for singing, and report on a user study on gamified singing. Through these applications, we demonstrate how advanced pitch estimation algorithms enhance musical interaction, education, and user experience in practical scenarios.

Finally, **Chapter 10** summarizes the thesis, reflects on its contributions to MIR and interactive music systems, and discusses potential avenues for future work. The importance of real-time capabilities in music technology is emphasized, encouraging continued exploration and innovation in this evolving field.

1.2 Own Publications and Contributions

This section lists the publications related to this thesis, all of which involve multiple authors. For each publication, my contributions as first author and main contributor are described, along with references to the corresponding parts of the thesis. These publications have been published in peer-reviewed journals and international conference proceedings, covering topics in MIR, audio signal processing, acoustics, informatics, and music computing. They have been published in or presented at internationally recognized journals and conferences, including TISMIR[4], ISMIR[5], DAFx[6], DAGA[7], GI[8], CMMR[9], ICMI[10], and SMC[11].

1.2.1 Publications with Peer-Review

The following peer-reviewed publications form the scientific foundation of this thesis and document its main research contributions.

[92] Peter Meier, Sebastian Strahl, Simon Schwär, Meinard Müller, and Stefan Balke. Pitch estimation in real time: Revisiting SWIPE with causal windowing. In *Proceedings of the International Symposium on Computer Music Multidisciplinary Research (CMMR)*, pages 285–297, London, UK, 2025. doi: 10.5281/zenodo.17496630.

This article forms the basis of Chapter 7 of this thesis. The first author, Peter Meier, is the main contributor to this article. In collaboration with Sebastian Strahl, Simon Schwär, Stefan Balke, and his supervisor Meinard Müller, he developed the ideas, designed the experiments, and wrote the paper. Peter Meier also developed and implemented all approaches and conducted the experiments. Sebastian Strahl contributed to the development and testing of the real-time implementation.

[90] Peter Meier, Meinard Müller, and Stefan Balke. A multi-user interface for real-time intonation monitoring in music ensembles. In *Proceedings of the Workshop for Innovative Computer-based Music Interfaces (ICMI)*, Chemnitz, Germany, 2025. doi: 10.18420/muc2025-mci-ws06-202.

This article forms the basis of Section 9.1 of this thesis. Peter Meier is the main contributor to this article. In collaboration with Stefan Balke and his supervisor Meinard Müller, he developed the ideas, designed the application, planned the experiments, and wrote the paper. Peter Meier also developed and implemented all approaches and conducted the user experiment. Stefan Balke contributed to the design and execution of the user experiment.

4 Transactions of the International Society for Music Information Retrieval (`https://transactions.ismir.net`)
5 International Society for Music Information Retrieval Conference (`https://ismir.net`)
6 International Conference on Digital Audio Effects (`https://dafx.de`)
7 Deutsche Jahrestagung für Akustik (`https://www.dega-akustik.de`)
8 Gesellschaft für Informatik e.V. (`https://gi.de`)
9 Computer Music Multidisciplinary Research Conference (`https://cmmr2025.prism.cnrs.fr`)
10 Workshop for Innovative Computer-based Music Interfaces (`https://www.icmi-workshop.org`)
11 Sound and Music Computing Conference (`https://smcnetwork.org`)

[91] Peter Meier, Meinard Müller, and Stefan Balke. Analyzing pitch estimation accuracy in cross-talk scenarios: A study with wind instruments. In *Proceedings of the Sound and Music Computing Conference (SMC)*, pages 3–10, Graz, Austria, 2025. doi: 10.5281/zenodo.15835032.

This article forms the basis of Chapter 8 of this thesis. The first author, Peter Meier, is the main contributor to this article. In collaboration with Stefan Balke and his supervisor Meinard Müller, he developed the ideas, designed the experiments, and wrote the paper. Furthermore, Peter Meier implemented all approaches and conducted the experiments.

[88] Peter Meier, Ching-Yu Chiu, and Meinard Müller. A real-time beat tracking system with zero latency and enhanced controllability. *Transactions of the International Society for Music Information Retrieval (TISMIR)*, 7(1):213–227, 2024. doi: 10.5334/tismir.189.

This article forms the basis of Chapter 3 of this thesis. The first author, Peter Meier, is the main contributor to this article. Together with Ching-Yu Chiu and his supervisor Meinard Müller, he developed the ideas, conducted the experiments, and wrote the paper. Furthermore, Peter Meier implemented the real-time beat tracking system and created the applications. Ching-Yu Chiu contributed her expertise on beat tracking and helped set up the datasets and experiments.

[89] Peter Meier, Simon Schwär, and Meinard Müller. A real-time approach for estimating pulse tracking parameters for beat-synchronous audio effects. In *Proceedings of the International Conference on Digital Audio Effects (DAFx)*, pages 314–321, Guildford, Surrey, UK, 2024.

This article forms the basis of Chapter 4 and Section 5.3 of this thesis. The first author, Peter Meier, is the main contributor to this article. Together with Simon Schwär and his supervisor Meinard Müller, he developed the ideas, formalized the mathematical description of the real-time pulse parameters, and wrote the paper. Furthermore, Peter Meier implemented the real-time approach and created a demo visualization. Simon Schwär developed an audio plugin and created mixing examples for the case studies.

[86] Peter Meier, Simon Schwär, Gerhard Krump, and Meinard Müller. Evaluating real-time pitch estimation algorithms for creative music game interaction. In *INFORMATIK 2023 – Designing Futures: Zukünfte gestalten*, pages 873–882. Gesellschaft für Informatik e.V., Bonn, Germany, 2023. doi: 10.18420/inf2023_97.

This article forms the basis of Chapter 6 and Section 2.2 of this thesis. The first author, Peter Meier, is the main contributor to this article. In collaboration with Simon Schwär and his supervisor Meinard Müller, he developed the ideas, designed the experiments, and wrote the paper. Furthermore, Peter Meier implemented all approaches and conducted the experiments. Simon Schwär contributed to the preparation of the dataset and developed a dataloader. Gerhard Krump was the supervisor of the main author at the Technische Hochschule Deggendorf.

[85] Peter Meier, Simon Schwär, Sebastian Rosenzweig, and Meinard Müller. Real-time MIR algorithms for music-reactive game world generation. In *Proceedings of the Workshop for Innovative Computer-based Music Interfaces (ICMI)*, Darmstadt, Germany, 2022. doi: 10.18420/muc2022-mci-ws03-225.

This article forms the basis of Section 5.2 of this thesis. The first author, Peter Meier, is the main contributor to this article. In collaboration with Simon Schwär, Sebastian Rosenzweig, and his supervisor Meinard Müller, he developed the ideas, designed the game application, and wrote the article. In addition, Peter Meier implemented all the approaches and created the music game prototype.

1.2.2 Publications without Peer-Review

The following works are related to this thesis but have not undergone formal peer review. They were presented at conferences, leading to some helpful feedback and discussion.

[87] Peter Meier, Simon Schwär, Gerhard Krump, and Meinard Müller. Real-time pitch estimation for creative music game interaction. In *Proceedings of the Deutsche Jahrestagung für Akustik (DAGA)*, pages 1346–1349, Hamburg, Germany, 2023.

This article forms the basis of Section 9.2 of this thesis. The first author, Peter Meier, is the main contributor to this article. In collaboration with Simon Schwär and his supervisor Meinard Müller, he developed the ideas, designed the application, and wrote the paper. In addition, Peter Meier implemented all approaches and created the music game prototype. Gerhard Krump was the supervisor of the main author at the Technische Hochschule Deggendorf.

[84] Peter Meier, Gerhard Krump, and Meinard Müller. A real-time beat tracking system based on predominant local pulse information. In *Demos and Late Breaking News of the International Society for Music Information Retrieval Conference (ISMIR)*, Online, 2021. URL `https://archives.ismir.net/ismir2021/latebreaking/000021.pdf`.

This article forms the basis of Section 5.1 of this thesis. The first author, Peter Meier, is the main contributor to this article. In collaboration with his supervisor Meinard Müller, he developed the ideas, designed the application, and wrote the paper. In addition, Peter Meier implemented all approaches and created the demo application system. Gerhard Krump was the supervisor of the main author at the Technische Hochschule Deggendorf.

1.3 Additional Publications

In addition to the main articles that form the core of this thesis, I also contributed to the following publication in the field of wind music research. This work is highly related to my research on cross-talk in wind music ensembles [91] and intonation monitoring [90].

[2] Stefan Balke, Peter Meier, and Meinard Müller. Practicing alone, playing together: A persona-based design approach for amateur wind musicians. In *Proceedings of the Workshop for Innovative Computer-based Music Interfaces (ICMI)*, Chemnitz, Germany, 2025. doi: 10.18420/muc2025-mci-ws06-199.

Stefan Balke is the main contributor to this article and authored the initial draft. Peter Meier, along with Meinard Müller and Stefan Balke, collaborated on discussing the concepts, developing the persona-based design approach, and writing the paper.

1.4 Acknowledgments

After almost five years, it is finally time to say thank you to all the people who have supported me throughout my Ph.D. journey. Without your help, encouragement, and inspiration, this thesis would not have been possible.

Before beginning my doctoral studies, I worked as an audio engineer in the Media Technology program at the Technische Hochschule Deggendorf (THD). In my daily work at the audio laboratory, I was involved in a wide range of projects, from listening tests, acoustic measurements, and music recordings to audio signal processing tasks. During this time, I learned about the BayWISS program, which supports researchers at Universities of Applied Sciences in Bavaria in pursuing a Ph.D. in cooperation with a university. This

sparked the idea of finding a research field and a supervisor who would allow me to combine my passion for audio engineering, programming, and music with my growing interest in research.

It was then that I discovered the work of Meinard Müller. As a musician, I was immediately impressed, not only by his research, including his book Fundamentals of Music Processing (FMP), but also by the way he openly shares knowledge through resources such as the FMP Notebooks, making music processing topics accessible and engaging for all. This marked my first encounter with the field of Music Information Retrieval (MIR). When I reached out to Meinard by email, he kindly invited me to a phone call. His openness, warmth, and enthusiasm for my ideas encouraged me to visit his group in Erlangen. Following that visit, we began collaborating on a small project, and eventually, I joined his group as an external Ph.D. student. Later, I became a full member of his group, working directly at AudioLabs in Erlangen.

Meinard, I am deeply grateful to you, for your exceptional mentorship, your unwavering support, and the trust you have placed in me. You have been an inspiring role model, combining scientific excellence with a deep passion for music, and you have always made time to listen, guide, and encourage. Your door was always open for my spontaneous questions, and I especially cherish our many "walk and talks," where ideas and advice flowed naturally. Beyond being an outstanding scientist, you have truly understood my situation as a father of two young children, showing deep empathy and consideration in supporting me to balance professional responsibilities with family life. Working with you has been not only a privilege but also a deeply human and inspiring experience, one that I will always treasure. I look forward to continuing our collaboration.

I would like to acknowledge my former supervisor at THD, Gerhard Krump († 2025), whose teaching and guidance in audio engineering and media technology supported my academic development and fostered my interest in these fields.

A great thank you goes to Christof Weiß for taking the time to read and review my thesis, for memorable hackdays in Würzburg, for making music together at Dagstuhl, and for our inspiring discussions about music during that time.

To my core research group at AudioLabs, "GroupMM", you have been more than just colleagues; you have been an inspiring, talented, and supportive scientific family. Working alongside you has taught me not only about research, technology, and music, but also about collaboration, generosity, and the joy of creating together. Thank you for sharing your expertise, your creativity, and your friendship: Jakob Abeßer, Vlora Arifi-Müller, Stefan Balke, Hans-Ulrich Berendes, Judith Bauer, Ching-Yu Chiu, Michael Krause, Ben Maman, Yigitcan Özer, Sebastian Rosenzweig, Abhirup Saha, Simon Schwär, Sebastian Strahl, Hendrik Schreiber, Christof Weiß, Frank Zalkow, and Johannes Zeitler.

Within GroupMM I had the pleasure to work with and supervise the Bachelor theses of Ole Müermann and Rico Rosenbusch. Thank you for the great discussions about music and beats.

My time at AudioLabs was further enriched by the wider community of researchers, engineers, and the dedicated secretarial and administrative staff. Each of you has contributed to my Ph.D. journey in countless ways, through stimulating discussions, practical help, and simply by making the lab a welcoming and enjoyable place to work. I am especially grateful to the administrative team, whose efficiency, kindness, and behind-the-scenes efforts kept everything running smoothly and allowed me to focus on my research. Many thanks to Bernd Edler, Carmen Craciun, Day-See Riechmann, Elke Weiland, Emanuël Habets, Jürgen Herre, Mandy Garcia, Petra Neubarth, Stefan Turowski, and the entire AudioLabs team.

The International Audio Laboratories Erlangen are a joint institution of the Friedrich-Alexander-Universität Erlangen-Nürnberg (FAU) and the Fraunhofer Institute for Integrated Circuits (IIS). Located inside the Fraunhofer IIS building, this unique collaboration combines the academic excellence of FAU with the applied research expertise of Fraunhofer IIS, creating an exceptional place for research and innovation. I am especially grateful to Bernhard Grill and Frederik Nagel, as representatives of Fraunhofer IIS, and to all who have contributed to establishing and sustaining this outstanding environment. Our Ph.D. seminars at Fraunhofer-Forschungscampus Waischenfeld were always a special highlight for me; a time to exchange ideas in depth, connect with fellow researchers, and find inspiration in a focused yet relaxed setting.

I would also like to thank Christian Dittmar for his valuable advice and support in his role as research coordinator at Fraunhofer IIS, as well as for the great hackdays we shared in Erlangen.

I am also grateful to my colleagues and students at THD, who have made my work there both productive and enjoyable. Special thanks go to Kristin Seffer and Roland Zink for their key role in launching my Ph.D. journey, for building a graduate school at THD, inviting me to serve as the first Ph.D. mentor in the program, and supporting me in countless ways along the way.

From BayWISS itself, I warmly thank Karin Streker for her constant support and for organizing excellent events that brought Ph.D. students together, fostered valuable exchange, and created a strong sense of community. I am also grateful to my fellow BayWISS Ph.D. colleagues for the inspiring discussions and the enriching exchange of ideas we shared. Finally, I would like to thank Andreas Nüchter and Wolfgang Mauerer, in their roles as directors of the BayWISS Verbundkolleg Digitalisierung.

After moving from Deggendorf to Erlangen, at AudioLabs I was supported by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) under Grant No. 500643750 (MU 2686/15-1) within the LEARN project, which I gratefully acknowledge.

One of the most inspiring experiences of my Ph.D. was participating in the Dagstuhl Seminar 24302 *"Learning with Music Signals: Technology Meets Education"* (`https://www.dagstuhl.de/24302`) in 2024, a week of engaging discussions, creative exchange, and memorable moments of making music together. Many thanks to the Dagstuhl team for providing such a welcoming and stimulating environment for research and collaboration. And to all participants, thank you for the thoughtful conversations, inspiring ideas, and the shared joy of music.

My thanks also go to Emmanouil Benetos for kindly inviting me to give a seminar talk at the Centre for Digital Music (C4DM) at Queen Mary University of London, and to Enzo De Sena for making my participation at the DAFx conference at the University of Surrey in Guildford a truly memorable experience.

To all my friends and fellow musicians with whom I have had the joy of making and performing music over the years. Thank you for the inspiration, the creativity, and the shared moments of musical expression that have been a constant source of motivation and joy throughout this journey. Music lies at the very heart of this thesis, and your friendship and artistry have enriched it in countless ways.

A very special shout-out goes to my brother Tobias Meier, my true musical soulmate, with whom I share a lifetime of making music, including in our own band. He is the one who meets me on the same wavelength, where every note feels natural. He has supported this work with his wonderful bass playing and has always been there with advice, encouragement, and practical help. I look forward to many more moments of making music together in the future.

Special thanks also go to Michael Brendel, not only for his friendship and encouragement, but also for his generous support in offering valuable feedback along the way.

Finally, my deepest gratitude goes to my family: My wife, my children, my parents, and my siblings. Your endless patience, love, and understanding have been the foundation that carried me through this journey. You stood by me through long hours, busy weeks, and moments of doubt, and celebrated every small victory along the way. You have been my anchor in difficult times and my greatest joy in good ones. I could not have done this without you. Thank you from the bottom of my heart!

This journey has been as much about people as about research, and for that, I am endlessly grateful.

2 Fundamentals: Real-Time Music Information Retrieval

In this chapter, we introduce the technical foundation and mathematical notation that will be used throughout the thesis. We begin by describing the fundamentals of audio signals (Section 2.1), then examine the principles and constraints of real-time audio processing (Section 2.2), and finally discuss the evaluation metrics used to assess interactive music systems (Section 2.3), especially related to beat tracking and pitch estimation tasks. For a comprehensive introduction to the fundamentals of music processing, we refer to the textbook by Müller [93], whose notation is consistently adopted in this thesis.

2.1 Audio Signals

This section provides an overview of the fundamental properties of audio signals, focusing on their representation in both the time domain (Subsection 2.1.1) and frequency domain (Subsection 2.1.2).

2.1.1 Time Domain

The following paragraphs cover essential aspects of audio signals in the **time domain**, such as oscillation, interference, sampling, the sampling theorem, aliasing, and quantization. Together, these topics establish the basis for understanding how sound is represented, sampled, and processed in digital systems.

Oscillation All auditory experiences, from perceiving rhythm and melody to analyzing multilayered audio recordings, stem from a simple oscillatory motion: repetitive variations in air pressure over time. These fluctuations propagate as **sound waves** in the air and can be received by the human auditory system. Specifically, the eardrum vibrates in response to these oscillations, transmitting mechanical motion through the middle ear and ultimately converting it into electrical signals in the inner ear. The brain interprets these signals as sound, forming the basis for the perception of pitch, loudness, and timbre.

A simple **sinusoidal oscillation** represents the fundamental building block of sound: a pure tone with a single frequency, amplitude, and phase, as illustrated by two examples in Figure 2.1. Mathematically, the

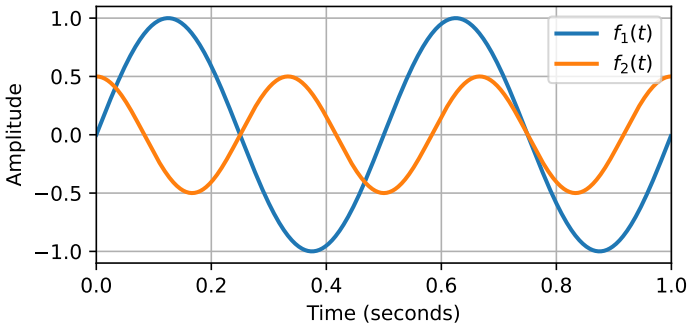


Figure 2.1: Two sinusoidal oscillations representing pure tones: $f_1 = 2\,\text{Hz}$, $A_1 = 1$, $\varphi_1 = 0$; $f_2 = 3\,\text{Hz}$, $A_2 = 0.5$, $\varphi_2 = \pi/2$. The x-axis shows time, the y-axis amplitude; frequency appears as cycles per second, and phase offset as a horizontal shift.

sinusoidal oscillation can be represented as a **continuous-time (CT) signal**, also referred to as an **analog signal**, $f : \mathbb{R} \to \mathbb{R}$, defined by the following equation

$$f(t) := A \cdot \sin(2\pi\omega t + \varphi), \tag{2.1}$$

where A is the **amplitude**, ω is the **frequency** in Hertz (Hz), t is **time** in seconds, and φ is the **phase** offset in radians. Although real-world musical sounds are usually much more complex, consisting of many oscillations combined, it is helpful to first understand the properties of a single sinusoidal wave. This simple model forms the basis for describing, synthesizing, and analyzing more complex audio signals in music.

Interference Interference describes the phenomenon that occurs when two or more audio signals are combined, resulting in a new signal whose amplitude at each point in time is determined by the sum of the individual signals. For example, consider two sinusoidal oscillations

$$f_1(t) = A_1 \cdot \sin(2\pi\omega_1 t + \varphi_1) \quad \text{and} \quad f_2(t) = A_2 \cdot \sin(2\pi\omega_2 t + \varphi_2),$$

with individual parameters for amplitude, frequency, and phase offset. The **superposition** of these two oscillations is given by

$$(f_1 + f_2)(t) := f_1(t) + f_2(t). \tag{2.2}$$

Depending on the relative phases φ_1, φ_2 and frequencies ω_1, ω_2, this can lead to **constructive interference** (amplitudes reinforce each other) or **destructive interference** (amplitudes partially or completely cancel each other out), as illustrated in Figure 2.2.

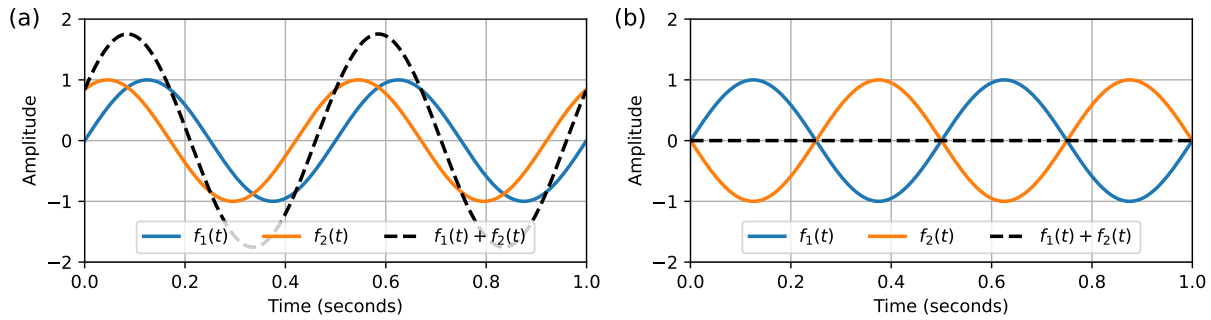


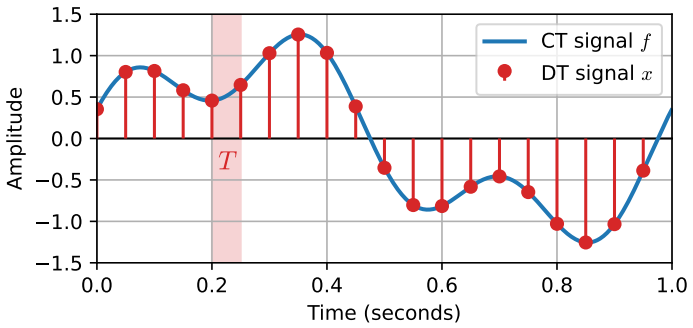
Figure 2.2: Constructive and destructive interference of two sine signals with identical frequency $\omega = 2\,\text{Hz}$ and amplitudes $A_1 = A_2 = 1$, but different phases. **(a)** Constructive interference with phases $\varphi_1 = 0$ and $\varphi_2 = 1$ (rad), resulting in amplitude amplification. **(b)** Destructive interference with phases $\varphi_1 = 0$ and $\varphi_2 = \pi$, causing cancellation of the signals.

Sampling Sampling is the process of converting a CT signal into a **discrete-time (DT) signal** by measuring its amplitude at regular time intervals, as illustrated in Figure 2.3. This process, known as **equidistant sampling**, transforms a CT signal $f\colon \mathbb{R} \to \mathbb{R}$ into a DT signal $x\colon \mathbb{Z} \to \mathbb{R}$ as given by the equation

$$x(n) := f(n \cdot T), \tag{2.3}$$

where n is the **sample index**, and T is the **sampling period**, which is the inverse of the **sampling rate** $F_\text{s} := 1/T$. The values $x(n)$ are called **samples** taken at time $t = n \cdot T$ of the original CT signal f.

Figure 2.3: Illustration of the sampling process, converting a CT signal f into a DT signal x by taking samples after regular intervals T. The x-axis represents time, and the y-axis represents amplitude.



Sampling Theorem A fundamental principle in digital signal processing is the sampling theorem (also known as the **Nyquist-Shannon sampling theorem**). It states that a CT signal f can be perfectly reconstructed from its samples $x(n)$ if the signal is Ω-bandlimited, meaning it contains no frequency components above a certain maximum frequency

$$\Omega := F_\text{s}/2. \tag{2.4}$$

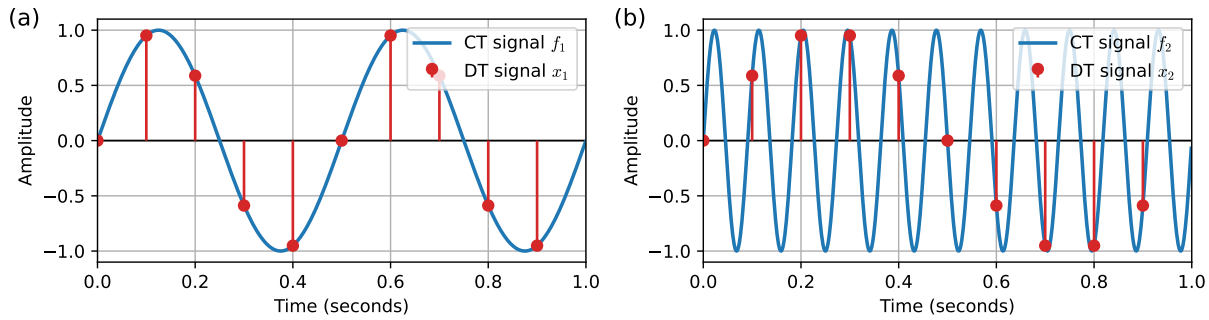


Figure 2.4: Sampling of two sine signals with frequencies $\omega_1 = 2\,\text{Hz}$ and $\omega_2 = 11\,\text{Hz}$, amplitudes $A_1 = A_2 = 1$, and phases $\varphi_1 = \varphi_2 = 0$. **(a)** With a sampling frequency of $F_s = 10\,\text{Hz}$, the signal of lower frequency, f_1, is accurately captured. **(b)** For the signal of higher frequency, f_2, the same sampling rate is insufficient, resulting in aliasing and a DT signal from which the CT signal cannot be perfectly reconstructed.

This bandlimit Ω is known as the **Nyquist frequency** and represents the highest frequency that can be accurately captured at a given sampling rate F_s. To ensure perfect reconstruction of the original signal, the sampling rate must be at least twice the maximum frequency present in the signal. If the signal contains frequency components above the Nyquist frequency, these components will be misrepresented in the sampled signal due to an effect called aliasing.

Aliasing If the sampling theorem is violated, meaning the sampling rate F_s is not sufficiently high to capture the maximum frequency of a signal, an effect called **aliasing** (from Latin *alias* meaning *other*) occurs, as illustrated in Figure 2.4.

To illustrate aliasing, consider sampling two sine signals f_1 and f_2 at a fixed sampling frequency of $F_s = 10\,\text{Hz}$. The first signal, f_1, has a frequency of $\omega_1 = 2\,\text{Hz}$, which is below the Nyquist frequency ($F_s/2 = 5\,\text{Hz}$). The second signal, f_2, has a frequency of $\omega_2 = 11\,\text{Hz}$, which exceeds the Nyquist limit by 6 Hz.

As shown in Figure 2.4a, f_1 is sampled appropriate, and its discrete representation accurately reflects the original oscillation. In contrast, for f_2, the sampling theorem is violated. Figure 2.4b demonstrates that the samples taken from f_2 do not capture its true high-frequency content. Instead, the sampled signal x_2 appears to oscillate at a much lower frequency (1 Hz), even though the original frequency was 11 Hz. This misrepresentation is the essence of aliasing: Frequencies above the Nyquist limit are reflected into the lower frequency range, producing aliases that cause ambiguity and loss of information in the sampled signal.

Quantization Quantization is the process of mapping the continuous range of amplitude values of a signal, taking values in $\mathbb{R}$, to a finite set of discrete levels $\Gamma \subset \mathbb{R}$. This is necessary because digital systems can only represent a limited number of distinct values, a quantity determined by the number of bits used for representation (referred to as the bit depth).

A quantizer is a system or algorithm that performs quantization. Mathematically, a quantizer can be described as a function $Q : \mathbb{R} \to \Gamma$ that assigns each real-valued amplitude $a \in \mathbb{R}$ to the nearest quantization level in Γ. A common type is the **uniform quantizer**, where the quantization levels are spaced evenly with a fixed step size Δ. The uniform quantization function is given by the equation

$$Q(a) := \operatorname{sgn}(a) \cdot \Delta \cdot \left\lfloor \frac{|a|}{\Delta} + \frac{1}{2} \right\rfloor, \tag{2.5}$$

where $\operatorname{sgn}(a)$ is the sign function (returning 1 for positive a, -1 for negative a, and 0 for $a = 0$), $|a|$ is the absolute value of a, Δ is the quantization step size (the distance between neighboring quantization levels), and $\lfloor \cdot \rfloor$ denotes the floor function, which rounds down to the nearest integer.

Quantization introduces a small error, called **quantization noise**, which is the difference between the original and quantized values. The smaller the step size Δ (i.e., the higher the bit depth), the lower the quantization error.

After the two steps of sampling (converting the CT signal to a DT signal) and quantization (mapping amplitudes to discrete levels), the original **analog** signal is now represented as a **digital** signal, suitable for storage, processing, and transmission in digital systems.

2.1.2 Frequency Domain

Having established how audio signals are represented and processed in the time domain (Subsection 2.1.1), we now turn to the **frequency domain**. While the time domain describes how a signal evolves over time, the frequency domain reveals the distribution and strength of its individual frequency components. The following paragraphs introduce various frequency domain concepts that are relevant to this thesis, including the **discrete Fourier transform (DFT)**, the **short-time Fourier transform (STFT)**, and the **spectrogram** as a visual representation of the STFT.

Discrete Fourier Transform (DFT) The DFT is a fundamental tool to analyze the frequency content of a real-valued DT audio signal $x \in \mathbb{R}^N$ of length $N \in \mathbb{N}$ samples. It transforms the time-domain signal x into complex-valued frequency coefficients $X \in \mathbb{C}^K$. We consider only the first $K = \left\lceil \frac{N+1}{2} \right\rceil$ coefficients, since the DFT of a real-valued signal is symmetric. The value K corresponds to the frequency index at the Nyquist frequency of the sampled input signal. Each coefficient represents the amplitude and phase of a specific frequency component present in the original signal. Following [93], we define the DFT as

$$X(k) := \sum_{n=0}^{N-1} x(n) \exp(-2\pi i k n / N), \tag{2.6}$$

where $x(n)$ denotes the n-th entry of the signal vector x, $k \in [0 : K-1]$ is the **frequency index**, and i represents the imaginary unit. In practice, the **fast Fourier transform (FFT)** is widely used as an efficient algorithm to compute the DFT.

Short-Time Fourier Transform (STFT) The STFT [46] provides a way to analyze how the frequency content of a signal changes over time by computing the Fourier transform on short, overlapping segments of the signal. Unlike the DFT, which analyzes the entire signal at once, the STFT slides a **window function** w of length $N \in \mathbb{N}$ across the signal in steps of $H \in \mathbb{N}$ samples (the **hop size**). For a DT signal x, the STFT $\mathcal{X}$ is defined as

$$\mathcal{X}(m,k) := \sum_{n=0}^{N-1} x(n+mH)w(n)\exp(-2\pi ikn/N). \tag{2.7}$$

Here, $\mathcal{X}(m,k)$ is the STFT coefficient at **time frame** $m \in [0 : M]$. For a signal of length L, the number of frames is $M := \left\lfloor \frac{L-N}{H} \right\rfloor$. The window function w selects a segment of N samples from the signal, starting at position $n = mH$. The parameter H, referred to as the hop size, specifies the number of samples by which the window is shifted for each successive frame. The choice of window function and its length N affects the frequency resolution, with longer windows providing finer frequency discrimination. The hop size H controls the time resolution, with smaller hop sizes offering finer temporal detail but increasing computational cost.

Physical Time and Frequency To relate the indices m and k in the STFT to physical time and frequency values, we use the following equations:

$$T_{\text{coef}}(m) := \frac{m \cdot H}{F_{\text{s}}} \tag{2.8}$$

$$F_{\text{coef}}(k) := \frac{k \cdot F_{\text{s}}}{N} \tag{2.9}$$

Here, the frame index m corresponds to the **physical time** position $T_{\text{coef}}(m)$ in seconds, and the frequency bin index k corresponds to the **physical frequency** $F_{\text{coef}}(k)$ in Hertz.

Spectrogram The spectrogram is a two-dimensional visualization that displays how the frequency content of a signal evolves over time, with one axis representing time, the other frequency, and color intensity indicating the magnitude or power of each frequency component. An example of a spectrogram is shown in Figure 2.5.

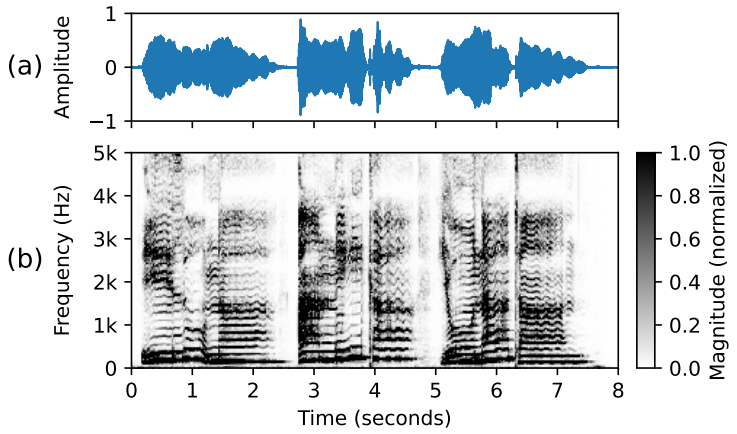


Figure 2.5: Spectrogram visualization of a vocal recording: **(a)** Time domain signal showing the amplitude of the audio waveform over time. **(b)** Spectrogram with logarithmic compression displaying the frequency content over time, where the x-axis represents time (in seconds), the y-axis represents frequency (in Hertz), and the color intensity indicates the normalized magnitude. Parameters: $N = 1024$, $H = 256$, $F_s = 22050$ Hz, and Hann window function w.

The equation below defines the spectrogram $\mathcal{Y}(m, k)$ as the squared magnitude of the STFT coefficients $\mathcal{X}(m, k)$, quantifying the energy present at each time frame m and frequency bin k:

$$\mathcal{Y}(m, k) := |\mathcal{X}(m, k)|^2. \tag{2.10}$$

For a more detailed discussion on the Fourier transform and digital signal processing, refer to [93, 98, 106].

Perceptual Pitch and Cent Humans perceive the fundamental frequency of a sound as **pitch**, which allows us to distinguish between higher and lower tones. While frequency in the physical domain is measured on a linear scale (Hertz), musical pitch perception is inherently logarithmic. This means that we perceive equal ratios of frequency as equal pitch intervals, rather than equal differences in frequency.

To bridge the gap between the physical (frequency) and perceptual (pitch) domains, music theory organizes pitch into discrete steps such as semitones and octaves. A common numerical representation is the **musical instrument digital interface (MIDI)** note number, where each integer $p \in [0, 127]$ corresponds to a specific pitch with center frequency

$$F_{\text{pitch}}(p) = 2^{(p-69)/12} \cdot 440, \tag{2.11}$$

where $p = 69$ denotes the standard pitch A4 (concert pitch) at 440 Hz. For fine-grained measurement of pitch intervals, the **cent** is used as a logarithmic unit. 1200 cents equal one octave, and 100 cents equal one semitone. The cent difference between two frequencies ω_1 and ω_2 is given by

$$1200 \cdot \log_2\left(\frac{\omega_1}{\omega_2}\right), \tag{2.12}$$

where a frequency ratio of 2 : 1 corresponds to 1200 cents. These pitch-based representations are fundamental in MIR tasks such as pitch detection, tuning analysis, and musical transcription, enabling a seamless link between physical acoustics and human perception.

2.2 Real-Time Audio Processing

> Parts of the this section are based on [86]. The first author, Peter Meier, is the main contributor to this article. See Chapter 6 for more details on the original publication and its contributions.

Most tasks in MIR research focus on analyzing large music datasets to extract information such as rhythm, melody, or harmony. These analyses are typically performed **offline**, where the entire audio signal is available for processing [127]. This allows for the use of advanced algorithms and large analysis windows, which can significantly improve accuracy compared to real-time methods. Offline methods generally prioritize analysis quality over execution time and assume unrestricted access to all data.

In contrast, **real-time** algorithms for interactive applications must operate under different constraints. Real-time processing requires algorithms to analyze audio as it arrives, with limited access to future data and strict timing requirements. Compared to offline processing, real-time methods for interactive applications differ in several important ways, including three key aspects outlined in [127].

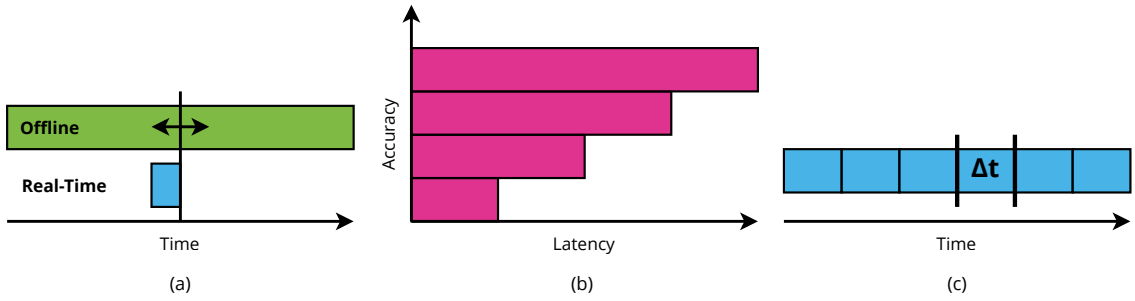


Figure 2.6: Illustration of three main differences between offline and real-time MIR. **(a)** Causality of information. **(b)** Trade-off between accuracy and latency. **(c)** Signal processing with blocks.

(a) **Causality of Information:** For offline analysis, the entire audio signal is already available, allowing movement forward or backward in the signal and selection of analysis windows of any size. For real-time analysis, only past and present information is available.

(b) **Trade-off Between Accuracy and Latency:** Generally, more available data enables more robust analysis (e.g., a DFT with more samples yields higher frequency resolution). However, in real-time analysis, this increases latency, as more samples lead to longer waiting times, especially for causal systems.

(c) **Signal Processing with Blocks:** Real-time analysis is performed in blocks, processing a fixed number of samples as provided by the soundcard. The analysis result is typically updated at the same block rate. Therefore, algorithms have limited time to complete before the next block. As a result, real-time methods often require reduced complexity compared to offline methods and must be carefully optimized to meet execution time constraints.

Addressing these challenges, the following sections discuss the principles of real-time audio processing that are essential for designing and implementing interactive music systems.

2.2.1 Audio Streaming

In Section 2.1, we introduced digital audio signals as the foundation for real-time audio processing. In this section, we focus on **audio streaming**, which is the continuous, real-time transmission and processing of digital audio data and therefore a key aspect of interactive music systems.

Signal Flow Audio streaming involves the flow of digital audio samples from an input source (e.g., microphone or instrument) through a processing pipeline and ultimately to an output destination (e.g., speakers or headphones). This process is typically managed by an **audio interface**, a hardware device that performs the conversion between analog and digital signal representations, as illustrated in Figure 2.7.

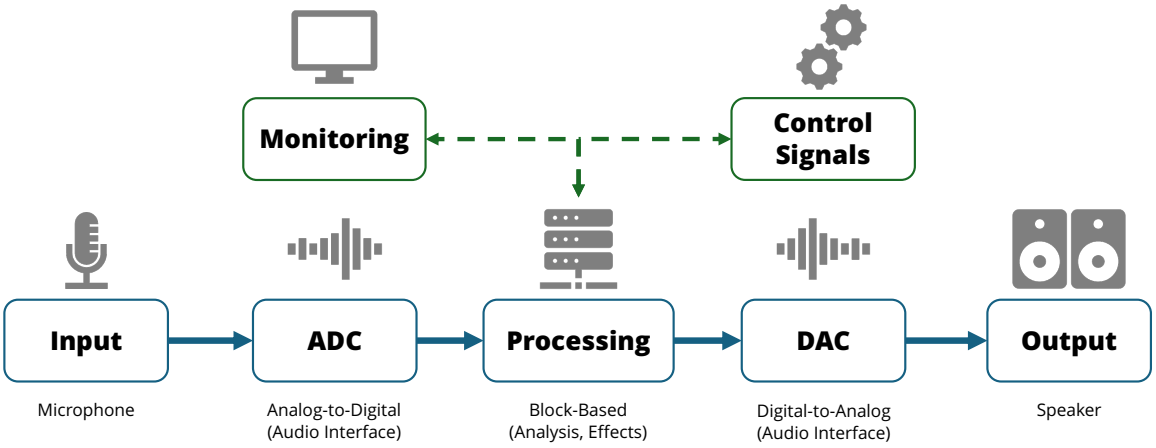


Figure 2.7: Signal flow in real-time audio streaming: Input sources are converted from analog to digital by the ADC, processed in blocks, converted back to analog by the DAC, and sent to output destinations. Monitoring data and control signals can be extracted during processing to analyze features or manipulate the audio stream in real time.

The audio interface samples the incoming CT signal at a fixed sample rate F_s, typically 44.1 kHz or 48 kHz for consumer audio, and up to 96 kHz or 192 kHz for professional applications, using an **analog-to-digital converter (ADC)**. Each sample is quantized to a specific bit depth, such as 16-bit for compact disc (CD) quality, 24-bit for professional recording, or 32-bit float for advanced audio processing. These digital samples are grouped into blocks, which are groups of samples processed together. Typical block sizes in

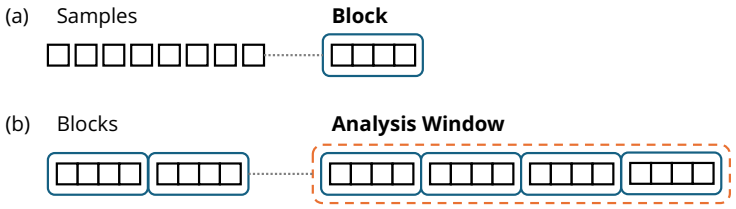


Figure 2.8: Buffering in real-time audio processing: **(a)** System-level buffering groups individual audio samples into blocks for processing. **(b)** Algorithmic buffering accumulates multiple blocks to form larger analysis windows required by many audio processing algorithms.

real-time audio systems are 32, 64, 128, 256, 512, or 1024 samples, commonly used as defaults in most audio hardware and software environments. This **block-based processing** can be used to analyze the audio signal to monitor various features or extract control signals. Conversely, these control signals can also be used to directly manipulate the audio stream within the same processing stage. After processing, the digital audio is converted back to analog by a **digital-to-analog converter (DAC)** in the audio interface and sent to the output destination, such as speakers or headphones.

Real-Time In audio streaming, **real-time** refers to the ability to process and output audio within the duration of a single block, ensuring immediate system response to user interactions or changes in the audio signal. The **block period** is the time available to process one block of audio samples, calculated as

$$T_{\text{block}} = \frac{B}{F_{\text{s}}}, \tag{2.13}$$

where B is the **block size** in samples (compare to the hop size H for the STFT) and F_{s} is the sampling rate.

Throughout this thesis, all algorithms and systems are designed and tested using a default block size B of 512 samples at a sample rate F_{s} of 44.1 kHz on consumer-level hardware. This configuration yields a block period T_{block} of approximately 11.6 milliseconds, which defines both the **latency** and the maximum allowable **processing time** per block for real-time operation.

2.2.2 Buffering and Causality

In audio streaming systems, different types of audio **buffering** are required to ensure reliable and efficient real-time processing. These buffering strategies address both **system-level** and **algorithmic** constraints, and will be discussed in the following paragraphs.

System-Level Buffering System-level buffering is responsible for grouping individual audio samples into **blocks** for processing, as illustrated in Figure 2.8a. Managed by the operating system (OS) or audio driver, this buffering temporarily stores incoming samples and delivers them as blocks to and from the audio interface. This mechanism regulates the flow of streaming audio data, helping the system absorb variations in processing time and preventing audible artifacts. The block size directly impacts system

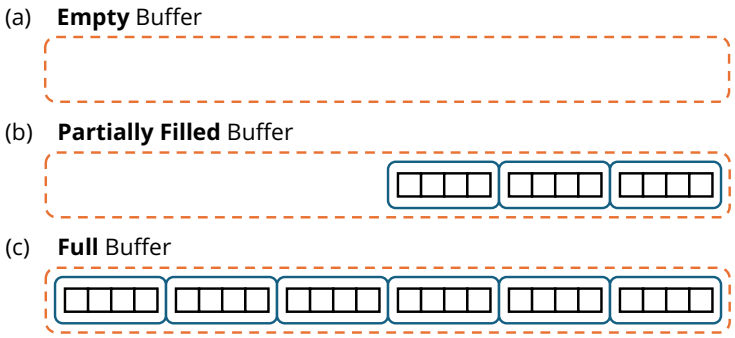


Figure 2.9: Buffer initialization effects in real-time audio processing: **(a)** At startup, an empty buffer leads to unreliable analysis. **(b)** As the buffer fills, analysis reliability improves. **(c)** Once full, the buffer enables robust and accurate analysis.

latency: Larger blocks provide more protection against sudden increases in central processing unit (CPU) load (e.g., temporary processing delays caused by other system tasks) but increase latency, while smaller blocks reduce latency but raise the risk of audio dropouts if blocks cannot be processed quickly enough.

Algorithmic Buffering Algorithmic buffering refers to the practice of accumulating multiple blocks to form larger **analysis windows** required by many audio processing algorithms, as illustrated in Figure 2.8b. While system-level buffering organizes samples into blocks for efficient input/output (I/O) operations between the audio interface and the system, algorithmic buffers are maintained internally by algorithms to provide sufficient temporal context for analysis tasks such as the STFT, onset detection, or beat tracking.

To illustrate the challenges of algorithmic processing, consider a simple case where an STFT uses a window size N of 2048 samples and a hop size H of 512 samples, with a block size B of 512 samples. In this setup, each STFT frame requires 4 consecutive blocks to form one analysis window, and one new block advances the window by one hop. In contrast, more complex algorithms may need much larger context windows. For example, a beat tracking algorithm may require around 3 seconds of audio to reliably detect rhythmic patterns and estimate tempo. At a sampling rate of 44.1 kHz and a block size of 512 samples, approximately 258 consecutive blocks are needed to fill this window. While such buffering enables robust analysis, it introduces trade-offs: larger buffers increase latency and memory usage. Therefore, buffering must be managed carefully to balance reliability with real-time constraints.

Buffer Initialization Effects Algorithmic buffers often exhibit **initialization effects** at the start of real-time audio streaming. As illustrated in Figure 2.9, the buffer is initially empty, resulting in no analysis. As blocks are received, the buffer fills and analysis reliability improves. Once fully populated, the buffer provides sufficient context for robust and accurate analysis. It is important to note that this initialization phase only affects the reliability of analysis at startup, not the system's responsiveness or latency. In practice, the initialization time is brief and has minimal impact on the overall user experience, as interactive music systems are typically used for extended periods.

Causality As a consequence of the signal flow used in audio streaming (see Figure 2.7) and the buffering strategies explained in the previous paragraphs, real-time audio processing is inherently **causal**. This means that only past and present audio data is available for the block-based processing, while future data is not accessible.

2.2.3 Real-Time Constraints

Callback Mechanism Unlike purely computation-driven processes, real-time systems rely on **callbacks**, where the audio interface hardware dictates when data is provided, and processing must be completed within a strict time frame to ensure uninterrupted audio streaming. This mechanism ensures that audio data is processed in a timely manner, allowing the system to maintain a continuous audio stream without interruptions.

```python
import sounddevice as sd

def callback(indata, outdata, frames, time, status):
    result = process(indata)  # Process the incoming audio data
    outdata[:] = result

# Start real-time audio stream
stream = sd.Stream(callback=callback, blocksize=512, samplerate=44100,
    channels=1)
```

Figure 2.10: Python example of a real-time audio stream using a callback function with the `sounddevice` library.

Figure 2.10 shows a Python example using the `sounddevice`[12] library to set up a real-time audio stream. The key concept is the `callback` function, which is automatically called by the audio interface each time a new block of audio data is available. The block size (`blocksize=512`) and sample rate (`samplerate=44100`) correspond to the values defined earlier in this section, meaning the callback is triggered every $T_{\text{block}} = (512/44100)\,\text{s} \approx 11.6\,\text{ms}$. Within the callback, the incoming audio (`indata`) is processed and the result is written to the output buffer (`outdata`). This structure ensures that audio processing is performed in sync with the hardware, maintaining a continuous stream of audio data.

Latency vs. Processing Time As a consequence of the callback mechanisms, real-time audio processing for each block must be completed before the next block arrives. This introduces a trade-off between **latency** and **processing time**: Reducing the block size decreases latency but increases the computational load and reduces the time available for processing each block.

Audible Artifacts If the processing time exceeds the block period T_{block}, a buffer underrun occurs, meaning there is not enough audio data available to play out in time. This leads to audible artifacts such

[12] `https://github.com/spatialaudio/python-sounddevice`

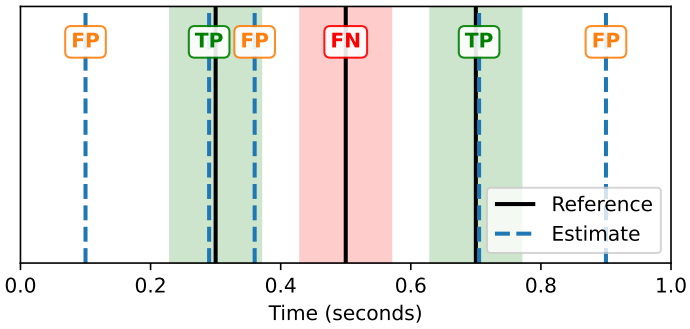


Figure 2.11: Example of beat tracking metrics. The figure shows reference beats (solid lines) and estimated beats (dashed lines) with tolerance windows highlighted. True positive (TP), false positive (FP), and false negative (FN) are annotated.

as clicks or pops in the output signal, an especially critical issue in real-time audio applications, where even a single missed sample can be perceptible. These artifacts typically manifest as short impulses in the time domain, which correspond to broadband noise in the frequency domain. To prevent such issues, it is essential that the processing for each block is always completed within the given block period.

2.3 Evaluation Metrics

To evaluate the real-time algorithms presented in this thesis, we employ a combination of commonly used statistical methods, task-specific accuracy metrics, and computational efficiency measures. The following subsections outline the evaluation approaches used for beat tracking (Subsection 2.3.1), pitch estimation (Subsection 2.3.2), and the performance evaluation of the algorithms in terms of real-time processing capabilities (Subsection 2.3.3).

2.3.1 Beat Tracking Metrics

Beat tracking algorithms are commonly evaluated using metrics such as **Precision**, **Recall**, and **F-measure**, which quantify the accuracy of estimated beats compared to annotated reference beat positions (ground truth) [32, 34]. An estimated beat is considered correct if it falls within a specified **time tolerance** window (typically ±70 ms) of a reference beat. The metrics are defined as follows:

- **Precision** (P): The proportion of estimated beats that are correctly matched to reference beats (i.e., how many detected beats are actually correct).
- **Recall** (R): The proportion of reference beats that are correctly detected by the algorithm (i.e., how many true beats were found).
- **F-measure** (F): The harmonic mean of Precision and Recall, providing a single accuracy score (that balances both, P and R). It is also known as **F1-score**, which we use in the following chapters.

These metrics are calculated using the following equations, where **true positive (TP)** are estimated beats that match reference beats within the time tolerance, **false positive (FP)** are estimated beats that do not match any reference beat, and **false negative (FN)** are reference beats that are not matched by any estimated beat, as illustrated in Figure 2.11:

$$P = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FP}} \tag{2.14}$$

$$R = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FN}} \tag{2.15}$$

$$F = 2 \cdot \frac{P \cdot R}{P + R} \tag{2.16}$$

Note that in Figure 2.11, the reference beat at 0.3 seconds contains two estimated beats within its tolerance window. Following the default implementation in Raffel et al. [107], only the closest estimated beat is matched to the reference beat and counted as a TP. Any additional estimated beats in the same window are treated as FP, as they do not correspond to unique reference beats. This approach enforces a one-to-one correspondence between reference and estimated beats, preventing over-counting.

For a detailed overview of additional beat tracking metrics we refer to [24, 32] and the tutorial by Davies et al. [34].

2.3.2 Pitch Estimation Metrics

In the context of pitch estimation, **voicing** refers to whether a frame of audio contains a pitched (voiced) sound or not. A frame is considered **voiced** if it contains a clear, periodic pitch (such as a sung note or a musical instrument tone), and **unvoiced** if it contains silence, noise, or non-pitched sounds. Voicing decisions are crucial because pitch estimation algorithms must not only estimate the correct pitch for voiced frames but also accurately identify unvoiced frames where no pitch should be reported. The most widely used metrics are **raw pitch accuracy (RPA)**, **raw chroma accuracy (RCA)** for pitch accuracy, and **voicing recall (VR)**, **voicing false alarm (VFA)** for voicing decisions [104, 107, 112]. Together, these metrics provide a comprehensive assessment of system performance across diverse musical contexts and signal conditions.

- **RPA**: Proportion of correctly estimated pitches (in cents) for frames where the reference is voiced.
- **RCA**: Proportion of correctly estimated pitch classes (ignoring octave errors) for voiced reference frames.
- **VR**: Proportion of voiced frames in the reference that are also estimated as voiced.

- **VFA**: Proportion of unvoiced frames in the reference that are incorrectly estimated as voiced.

Let M denote the total number of frames, m the frame index, v_{ref} and v_{est} the voicing indicator vectors (1 for voiced, 0 for unvoiced), and c_{ref} and c_{est} the reference and estimated pitches in cents. The metrics can be defined as follows:

$$\text{RPA} = \frac{\sum_{m=1}^{M} \delta(c_{\text{ref}}^m, c_{\text{est}}^m) \cdot v_{\text{ref}}^m}{\sum_{m=1}^{M} v_{\text{ref}}^m} \tag{2.17}$$

$$\text{RCA} = \frac{\sum_{m=1}^{M} \delta_{\text{chroma}}(c_{\text{ref}}^m, c_{\text{est}}^m) \cdot v_{\text{ref}}^m}{\sum_{m=1}^{M} v_{\text{ref}}^m} \tag{2.18}$$

$$\text{VR} = \frac{\sum_{m=1}^{M} v_{\text{ref}}^m \cdot v_{\text{est}}^m}{\sum_{m=1}^{M} v_{\text{ref}}^m} \tag{2.19}$$

$$\text{VFA} = \frac{\sum_{m=1}^{M} (1 - v_{\text{ref}}^m) \cdot v_{\text{est}}^m}{\sum_{m=1}^{M} (1 - v_{\text{ref}}^m)} \tag{2.20}$$

Here, $\delta(a, b)$ is an indicator function that determines whether the estimated pitch is sufficiently close to the reference pitch. It is defined as:

$$\delta(a, b) := \begin{cases} 1, & \text{if } |a - b| < C \\ 0, & \text{otherwise,} \end{cases} \tag{2.21}$$

where C is the cent tolerance (e.g., 50 cents). In other words, $\delta(a, b)$ returns 1 if the absolute difference between the estimated and reference pitch is less than C, and 0 otherwise. For chroma accuracy, which ignores octave errors, the indicator function is modified to operate on pitch classes:

$$\delta_{\text{chroma}}(a, b) := \begin{cases} 1, & \text{if } \left| (a - b) - 1200 \cdot \text{round}\left(\frac{a-b}{1200}\right) \right| < C \\ 0, & \text{otherwise} \end{cases} \tag{2.22}$$

Here, the difference $a - b$ is mapped to the closest multiple of 1200 cents (one octave), so octave errors are ignored and only the pitch class distance is considered.

As a practical example of RPA, we consider a sequence of reference and estimated pitches, as shown in Figure 2.12. The reference pitch values are given in cents, with a tolerance of ±50 cents for correct estimates. The estimated pitch values are also in cents, and the voicing indicator vector v_{ref} indicates which reference frames are voiced (1) or unvoiced (0). The figure illustrates how the RPA metric is calculated by checking if the estimated pitch falls within the tolerance window of the reference pitch for each voiced

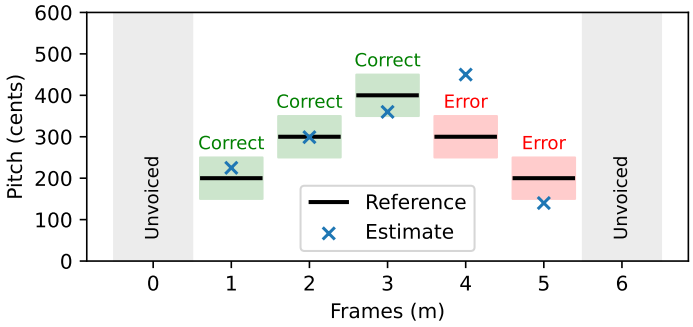


Figure 2.12: RPA visualization: Reference and estimated pitch values are displayed per frame index. Green boxes mark the tolerance window (±50 cents) for correct estimates, while red boxes show incorrect estimates outside the tolerance. Unvoiced reference frames are shaded gray.

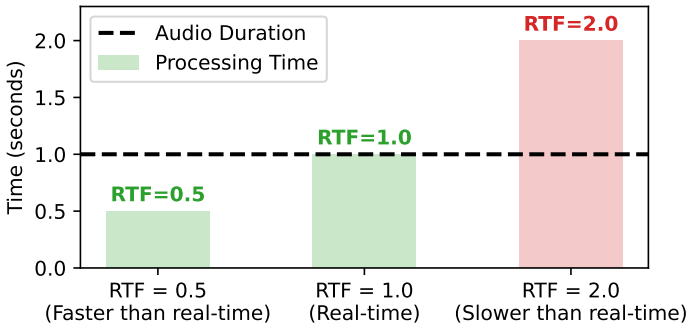


Figure 2.13: Audio duration and processing time in seconds. For real-time processing, the processing time should be less than or equal to the audio duration. Green boxes show real-time processing; red boxes indicate slower-than-real-time processing.

frame. Correct estimates are highlighted in green, while incorrect estimates are shown in red. With this, the raw pitch accuracy is calculated as RPA = 3/5 = 0.6.

2.3.3 Performance Evaluation

The performance of real-time algorithms is typically evaluated using the **real-time factor (RTF)**, which is defined as the ratio of the processing time required to analyze a given audio segment to the duration of that segment. The RTF is calculated as follows:

$$\text{RTF} = \frac{\text{Processing Time}}{\text{Audio Duration}} \tag{2.23}$$

Figure 2.13 illustrates the concept of RTF with a simple example. An RTF of 1.0 means the algorithm processes audio in real time: the processing time equals the audio duration. If the RTF is greater than 1.0, the algorithm is slower than real time, taking longer to process the audio than its duration. Conversely, an RTF less than 1.0 indicates processing faster than real time, which is desirable for real-time applications. For instance, an RTF of 0.5 means the algorithm processes audio twice as fast as real time, while an RTF of 2.0 means processing takes twice as long as the audio duration.

Part I

Real-Time Beat Tracking

3 A Real-Time Approach for Predominant Local Pulse Estimation

This chapter is based on [88]. The first author, Peter Meier, is the main contributor to this article. Together with Ching-Yu Chiu and his supervisor Meinard Müller, he developed the ideas, conducted the experiments, and wrote the paper. Furthermore, Peter Meier implemented the real-time beat tracking system and created the applications. Ching-Yu Chiu contributed her expertise on beat tracking and helped set up the datasets and experiments.

Identifying beat positions in music recordings, a central task in MIR, is commonly referred to as beat tracking. Typically, this involves computing an activation function to reveal frame-wise beat likelihood and then conducting post-processing to derive final beat positions. Existing methods often operate offline, requiring access to the entire music track for processing. In this chapter, we introduce a real-time beat tracking system based on the PLP concept, originally designed for offline use. Our main contribution is the successful transformation of the PLP-based algorithm into a real-time procedure. Unlike traditional offline methods providing static beat positions, our real-time approach dynamically captures changes in local pulse characteristics with each frame of an audio stream. This yields additional insights, including beat context, beat stability, and beat lookahead for predicting beats in advance. In this way, our system not only demonstrates high controllability for real-time applications but also can operate at zero latency. Additionally, we present experiments comparing our real-time beat tracking system with other models and evaluating the accuracy of our lookahead feature. In summary, our real-time beat tracking system offers a lightweight algorithm that is particularly well-suited for interactive music software development. The subsequent sections of this chapter follow this structure: We begin in Section 3.1 with an overview of the topic and our approach. Section 3.2 explores the original PLP-based algorithm. Section 3.3 discusses its conversion into a real-time procedure. In Section 3.4, we analyze the resulting system output. Section 3.5 presents multiple experiments, and Section 3.6 discusses their results. Finally, Section 3.7 concludes our method. In Chapter 5, we demonstrate our real-time system through real-world beat tracking applications. Additional materials and audio examples are available on a supplemental website.[1]

[1] https://audiolabs-erlangen.de/resources/MIR/2024-TISMIR-RealTimePLP

3.1 Introduction

Beat tracking is a fundamental task in MIR, forming a core step in the analysis and understanding of musical content. The general idea of beat tracking is to have a system that analyzes music and automatically taps along with the beat, similar to what human listeners would do [37, 41]. The human perception of beats in music can occur at different metric levels, for example: the tactus level (quarter notes), the measure level (bars), and the tatum level (temporal atom), which refers to the fastest repeating temporal pattern in the music [93].

Besides beat tracking, there are several closely related concepts: While tapping and marking beat positions in a piece of music, beats can be counted and grouped into bars, a process called **meter tracking** [62, 69]. Identifying the first beat of each bar is helpful for many music applications and is referred to as **downbeat tracking** [12, 40, 75]. Analyzing the number of beats per minute (BPM) helps to identify the tempo of the music, which is commonly known as **tempo tracking** [74].

In this work, we mainly focus on the task of beat tracking that involves identifying the time positions of beats within a music recording and is essential for MIR applications ranging from music transcription to rhythm-based interactive systems. Typically, beat tracking is a two-step process: first, calculating an activation function to extract frame-wise beat likelihood from the music signal, and second, performing a post-processing method to determine final beat positions from the activation function.

Most existing beat tracking methods are designed for offline use and rely on access to the entire music track for processing. In recent years, deep learning (DL) has led to substantial improvements in beat tracking with approaches such as temporal convolutional networks (`TCN`) [8, 30], `Transformer` [139], or `SpecTNT-TCN` [63]. With the growing demand for real-time applications like interactive music systems and live performance tools, the need for online-capable beat tracking algorithms has become more evident. This trend is reflected in contributions such as `BeatNet` [59], `Novel-1D` [60], or the most recent model `BEAST-1` [23].

When employing online approaches for real-time applications or integrating them into larger interactive systems, two prominent challenges emerge: latency and controllability. First, latency encompasses delays not only from the beat tracker itself but also from network communication, audio output processing, or controller input lags. These delays make it challenging to maintain synchronization between analyzed input audio streams and generated output audio streams. Second, controllability in this context refers to the challenges most online beat trackers face in being easily adjustable for specific real-time application requirements. Most real-time beat tracking systems lack explicit control over parameters such as pulse level (e.g., quarter-note or eighth-note level) or the amount of latency.

In this chapter, we present a real-time beat tracking system built on the PLP concept [53], originally developed for offline use. Our main contribution is the successful transformation of the PLP-based algorithm into a real-time procedure. With our approach, we achieve beat detection performance

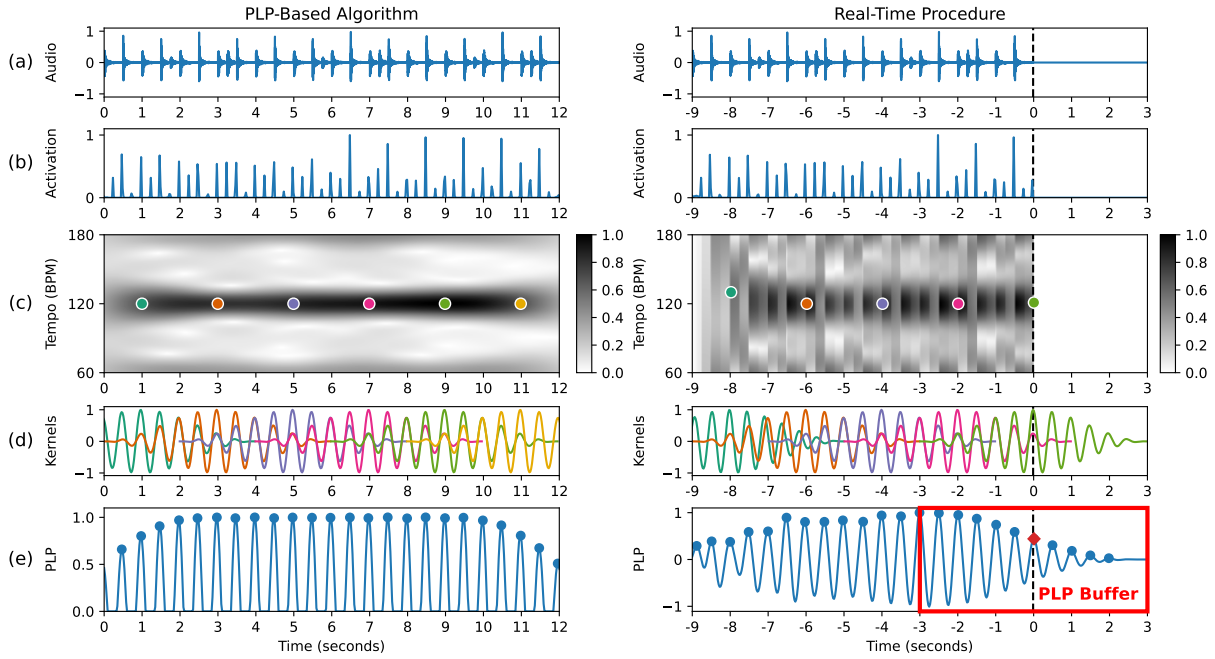


Figure 3.1: Illustration of the original offline PLP-based algorithm (left side), as described in Section 3.2, and the real-time procedure (right side), detailed in Section 3.3. **(a)** Audio signal. **(b)** Activation function. **(c)** Tempogram. **(d)** Pulse kernels. **(e)** PLP function. To provide clearer visualization and illustrate the general idea, we plot kernels only at 2-second intervals.

comparable to that of most other online beat trackers. In contrast to previous beat trackers, our method allows for achieving zero latency while remaining lightweight and easily controllable. In particular, our approach incorporates beat lookahead functionality to mitigate latency effects in interactive real-time systems and provides enhanced controllability through adjustable parameters.

To evaluate the influence of the lookahead feature on beat detection performance, we conduct experiments using a variety of datasets. In addition to quantitative evaluation, we demonstrate the practical usability of our system through two real-world applications for interactive music making and educational music gaming (see Chapter 5). These applications creatively utilize the output of our system, highlighting its controllability and versatility. In summary, our real-time beat tracking system represents a lightweight algorithm that is especially useful for interactive music software development. By bridging the gap between offline beat tracking methods and real-time application requirements, our system opens up new possibilities for musicians to use real-time beat tracking for their interactive music systems.

3.2 PLP-Based Algorithm

Before discussing how to convert the PLP-based algorithm into a real-time procedure (Section 3.3), we first look at the original offline procedure [53]. All the essential steps for calculating the PLP are illustrated in Figure 3.1 (left column).

First, the audio signal (Figure 3.1a) is transformed into an activation[2] function $\Delta : \mathbb{Z} \to \mathbb{R}$ for time positions $n \in \mathbb{Z}$. The peaks of the activation function Δ (Figure 3.1b) indicate the likelihood of observing a beat at time positions n. To identify local periodic patterns, we perform a STFT on Δ. For this, we define a window function $\mathcal{W} : [-N : N] \to \mathbb{R}$ for $N \in \mathbb{N}$ that is normalized and centered around each time position n, resulting in a total window size of $K = 2N + 1$. For an arbitrary but fixed time position $n \in \mathbb{Z}$, the window $\mathcal{W}$ defines a neighborhood indexed by $m \in [n - N : n + N]$. The complex-valued Fourier coefficient

$$\mathcal{F}(n, \omega) = \sum_{m=n-N}^{n+N} \Delta(m) \cdot \mathcal{W}(m - n) \cdot \exp(-2\pi i \omega m) \tag{3.1}$$

obtained from the STFT, is defined for time positions $n \in \mathbb{Z}$, frequencies $\omega \in \mathbb{R}_{\geq 0}$, and local time indices $m \in [n - N : n + N]$. From the Fourier coefficient $\mathcal{F}(n, \omega)$ we derive a Fourier tempogram $\mathcal{T}(n, \tau)$, given by

$$\mathcal{T}(n, \tau) = \big|\mathcal{F}(n, \tau/60)\big|. \tag{3.2}$$

This time–frequency representation of the activation function Δ, as illustrated in Figure 3.1c, is commonly measured in BPM rather than Hertz, with a tempo parameter $\tau = 60 \cdot \omega$. Utilizing the Fourier tempogram $\mathcal{T}(n, \tau)$ relies on two assumptions commonly employed in beat tracking. First, beat positions go along with note onsets, and, second, beat positions are periodically spaced. These assumptions are exploited by comparing local sections of the activation function Δ with windowed sinusoidal kernels $\kappa_n : [n - N : n + N] \to \mathbb{R}$, given by the equation

$$\kappa_n(m) := \mathcal{W}(m - n) \cdot \cos\Big(2\pi\big((\tau_n/60) \cdot m - \varphi_n\big)\Big), \tag{3.3}$$

for $m \in [n - N : n + N]$, as shown in Figure 3.1d. To obtain a local beat (or pulse) tracker, we choose for each time position $n \in \mathbb{Z}$ a kernel κ_n that optimally matches the local tempo structure of the signal within a given tempo range Θ, as illustrated with colored dots in Figure 3.1c. For instance, when denoting $\Theta = [30 : 240]$, we refer to a tempo range spanning from 30 to 240 BPM. In Equation 3.3, τ_n is defined as the tempo parameter that maximizes the tempogram $\mathcal{T}(n, \tau)$ for each time position n.

$$\tau_n := \operatorname*{argmax}_{\tau \in \Theta}\big(\mathcal{T}(n, \tau)\big) \tag{3.4}$$

Additionally, φ_n represents the phase of the windowed sinusoid with tempo τ_n that best correlates with the local section around n of the activation function Δ.

$$\varphi_n = -\frac{1}{2\pi}\operatorname{angle}\big(\mathcal{F}(n, \tau_n/60)\big) \tag{3.5}$$

[2] Note that the input to the PLP method can include both novelty functions (such as spectral flux) and activation functions (such as the output of a recurrent neural network (RNN)).

To obtain a globally defined pulse tracker, the final step of the original offline procedure involves overlap-adding all optimal pulse kernels over time to form a global pulse function $\Gamma : \mathbb{Z} \to \mathbb{R}_{\geq 0}$:

$$\Gamma(n) = \left| \frac{1}{C} \sum_{\ell=n-N}^{n+N} \kappa_\ell(n) \right|_{\geq 0} \tag{3.6}$$

This curve, which reveals predominant local pulse information, is referred to as the PLP curve. To reach its final state, the PLP function Γ undergoes two additional computation steps, included in Equation 3.6: normalization and half-wave rectification. The constant

$$C = \sum_{n=-N}^{N} \mathcal{W}(n) \tag{3.7}$$

ensures the normalization of the PLP function, keeping the values $\Gamma(n)$ within the range of $[-1 : 1]$. With half-wave rectification, we only consider the positive values of the PLP function, where $|x|_{\geq 0} := x$ for a non-negative real number x and $|x|_{\geq 0} := 0$ for a negative number x. For a more detailed description, we refer to [53].

3.3 Real-Time Procedure

MIR is an area of research that often deals with the analysis of entire corpora. The algorithms used for these analyses typically operate offline. In contrast to this, the algorithms used for real-time analysis face multiple challenges. This includes working with causal data only, addressing the trade-off between accuracy and latency, and meeting audio processing deadlines, as discussed by [127]. In adapting our model from offline to real-time (Subsection 3.3.1), we encountered three significant differences, which we discuss in more detail in the following subsections. First, data is streamed in small blocks, requiring the balancing of latency and computation times (Subsection 3.3.2). Second, only causal data is available, affecting the overall accuracy of the beat tracker (Subsection 3.3.3). Third, data needs to be stored in buffers, enabling access to previous time frames for handling larger context windows (Subsection 3.3.4).

3.3.1 Real-Time PLP Algorithm

The original offline procedure outlined in Section 3.2 can be transformed into a real-time procedure, as depicted in Figure 3.1 (right column). Given the centric nature of the kernels κ_n around each time position n, the kernel window $\mathcal{W}$ is essentially divided into two halves, see Figure 3.1d. The left half of the kernel window supporting $[-N : 0]$ is utilized to compute the pulse structure based on past and present data, while the right half supporting $(0 : N]$ is used to extrapolate this pulse structure to future time positions. This allows the superimposed kernels κ_n to predict future beat positions, as illustrated in Figure 3.1e.

To describe the real-time procedure, we update the equations in Section 3.2 as follows. Let n_0 denote the current time position, where we have access to activation values $\Delta(n)$ for all time positions $n \leq n_0$. The complex-valued Fourier coefficient $\mathcal{F}(n, \omega)$ from Equation 3.1 becomes

$$\mathcal{F}'(n, \omega) = \sum_{m=n-N}^{n} \Delta(m) \cdot \mathcal{W}(m - n) \cdot \exp(-2\pi i \omega m), \tag{3.8}$$

which is defined for all time positions $n \leq n_0$, frequencies $\omega \in \mathbb{R}_{\geq 0}$, and local time indices $m \in [n - N : n]$. For the current time position n_0, we obtain a real-time PLP function called $\Gamma_{n_0} : [-\infty : n_0 + N] \to \mathbb{R}$

$$\Gamma_{n_0}(n) := \frac{1}{C} \sum_{\ell=n-N}^{n_0} \kappa_\ell(n), \tag{3.9}$$

which is defined for all time positions $n \in [-\infty : n_0 + N]$ and has access to all kernels κ_ℓ for $\ell \in [n - N : n_0]$. For normalization, we use the constant C from Equation 3.7, which ensures that the values $\Gamma_{n_0}(n)$ lie within the range of $[-1 : 1]$. Additionally, for the real-time PLP function, we choose to skip half-wave rectification and preserve a more sinusoidal state to better capture dynamic pulse changes over time. Note that the kernels κ_ℓ are computed based on $\mathcal{F}'(n, \omega)$ (Equation 3.8), denoted with $\mathcal{T}'(n, \tau)$ for the Fourier tempogram (Equation 3.2), τ_n' for the tempo parameter (Equation 3.4), and φ_n' for the phase (Equation 3.5). The PLP buffer, depicted in Figure 3.1e, displays only the section of $\Gamma_{n_0}(n)$ for $n \in [n_0 - N : n_0 + N]$, containing all the necessary information to compute the subsequent time position n_1. Note that for this buffer, our perspective shifts from a linear time scale to a centric viewpoint, where the center of the buffer represents the current time position n_0 (which corresponds to the physical buffer time position $t = 0$, as illustrated in Figure 3.2).

3.3.2 Audio Streaming

The main difference between offline and real-time processing lies in how data is delivered. In offline processing, the entire audio track is available at any time. Conversely, in real-time processing, audio data is streamed continuously as it becomes accessible. Streamed audio is commonly processed in small **blocks** of data known as **frames**, which directly correspond to the time positions n introduced in Section 3.2. Each frame contains a fixed frame size $B \in \mathbb{N}$ of recent audio data sampled at a fixed audio sampling rate F_s. The duration of a single frame is called frame period T, as given by the equation

$$T = \frac{B}{F_s}. \tag{3.10}$$

Consequently, when we receive a frame of data, we are inherently operating with a **latency** equivalent to one frame period T. For example, if a real-time system runs with an audio sampling rate $F_s = 44100$ Hz and a frame size $B = 512$ samples, the system latency based on the frame period is $T = 512/44100 = 11.61$

ms. Reducing the frame size B to minimize system latency is often not feasible because the frame period T also determines the available computation time per frame.

3.3.3 Causal Data Processing

PLP operates with centric kernels κ_n, analyzing a specific time position n within a larger context window $\mathcal{W}$ by utilizing both past and future data, with local time indices $m \in [n - N : n + N]$. However, for the real-time procedure, only causal data is available, limiting the indices to $m \in [n - N : n]$. While delaying the computation to await future data is possible, this approach is not feasible for real-time applications. For this reason, in our real-time model, we only utilize past and present data for PLP computation. This helps avoid additional latency but also impacts other aspects of the beat tracking pipeline, as we will discuss in the following.

The tempogram $\mathcal{T}'(n, \tau)$ is based on only past and present activation data, which has multiple effects. First, the data for calculation is essentially half the size and therefore less accurate. Second, the lack of future data makes it harder to adapt to upcoming tempo changes. Third, the activation data displays a discontinuity, abruptly dropping to zero due to the absence of future data. This discontinuity introduces artifacts, manifesting as vertical lines in the tempogram, as depicted in Figure 3.1c.

The real-time PLP function Γ_{n_0} incorporates only past and present kernels κ_ℓ, as we restrict the kernel summation in Equation 3.9 to local time indices $\ell \in [n - N : n_0]$. Consequently, the right half of the PLP buffer, which represents future pulse data, consistently exhibits a falling slope due to the absence of overlapping kernels on that side of the buffer. This leads to less prominent peaks in the PLP buffer $\Gamma_{n_0}(n)$ for $n \in [n_0 - N : n_0 + N]$ characterized by overall lower amplitude values, as illustrated in Figure 3.1e.

3.3.4 Data Buffering

Compared to a single frame size B, the kernel window $\mathcal{W}$ for PLP is relatively long, typically utilizing kernel sizes K of $4 - 12$ s. Therefore, a single frame of audio data is insufficient for computing PLP, and it is necessary to collect and store frames in a buffer. To achieve this in a memory-efficient manner, we buffer only as much data as necessary to compute the current frame. For this purpose, a First-In-First-Out circular buffer is utilized, where the oldest values are dropped as new values are added. For real-time PLP, there are two computations where buffers are needed. First, to compute a kernel, an **activation buffer** with half kernel size K is required ($K_{\text{Act}} = N + 1$), supporting $[-N : 0]$. Second, to overlap-add kernels with previous kernels, a **PLP buffer** with full kernel size K is needed ($K_{\text{PLP}} = K = 2N + 1$), supporting $[-N : N]$. The PLP buffer is fully described by $\Gamma_{n_0}(n)$ for the section $n \in [n_0 - N : n_0 + N]$ in Equation 3.9.

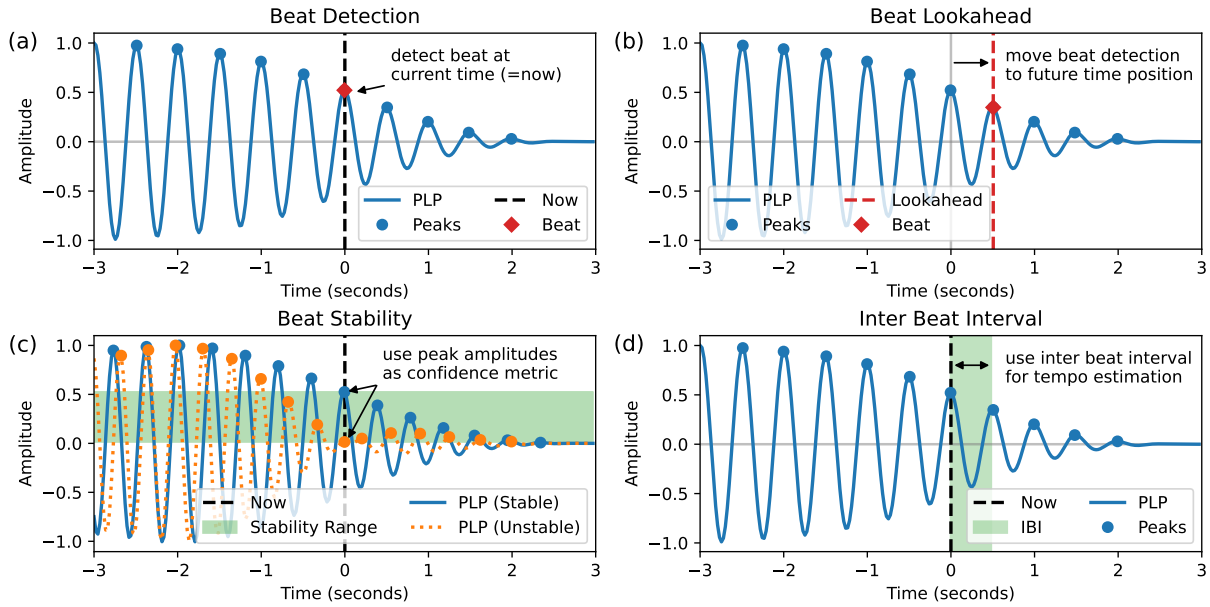


Figure 3.2: The output of the real-time beat tracking system: **(a)** Beat Detection (Subsection 3.4.1). **(b)** Beat Lookahead (Subsection 3.4.3). **(c)** Beat Stability (Subsection 3.4.4). **(d)** Inter Beat Interval (Subsection 3.4.5).

3.4 System Output

The primary objective of the real-time procedure (Section 3.3) is to update the PLP buffer with every new frame of data, thereby generating all the system output, as we will discuss in the following subsections.

3.4.1 Beat Detection

With Figure 3.2a, we demonstrate the process of **beat detection** in our real-time beat tracking system. For offline processing, the goal of beat detection is to compile a list of beat positions, indicating when beats occur relative to the start of an audio track. For real-time audio streaming, the objective of beat detection is to determine if a beat should be triggered at the current time position n_0. This is achieved by analyzing the current state of the PLP buffer and using a simple peak picking method to identify peak positions $P = \{p_1, p_2, \ldots\}$ within the section $n \in [n_0 - N : n_0 + N]$. If a peak $p_i \in P$ falls at the center time position of the PLP buffer ($p_i = n_0$), a beat occurs at the current time position n_0 and should be triggered immediately.

3.4.2 Beat Context

Even though the beat detection method outlined in Subsection 3.4.1 primarily focuses on the center time position n_0 of the PLP buffer, every peak $p_i \in P$ provides additional valuable information. They create a form of **beat context** around the current time position n_0, giving insights into both past and potential

future beat positions. This functionality proves useful for real-time applications, as demonstrated in Section 5.2. The extent to which past and future information is available can be adjusted by modifying the PLP kernel size K. However, this also affects how sensitive the beat tracker is to tempo variations in the music: Increasing the kernel size K reduces the tracker's sensitivity to tempo fluctuations. Conversely, decreasing the kernel size K enhances the tracker's responsiveness to tempo changes.

3.4.3 Beat Lookahead

For beat detection, as discussed in Subsection 3.4.1, we typically use the center position n_0 of the PLP buffer as the "decision line." However, by considering the beat context (Subsection 3.4.2), we can shift this decision line to any time position $n \in [n_0 - N : n_0 + N]$, including future time positions $n > n_0$, as illustrated in Figure 3.2b. To achieve this, we introduce the **beat lookahead**, an input parameter influencing the real-time beat tracking system's behavior. Instead of detecting beats at the center time position n_0 of the PLP buffer, the beat lookahead defines an offset, to move the decision line by a fixed number of frames to detect and trigger beats ahead of time.

Although triggering beats earlier may not appear intuitive, it becomes a valuable solution in addressing significant latency issues inherent in most real-time applications. These delays often arise from various sources, including audio processing, network communication, or input controller lags. Therefore, the beat lookahead is crucial for compensating for these latencies and ensuring synchronization between the analyzed input signal and the generated output signal.

However, predicting beats ahead of time comes with a trade-off: the accuracy of beat detection diminishes to some extent, which we elaborate on in Subsection 3.6.4.

3.4.4 Beat Stability

Every time beat detection (Subsection 3.4.1) occurs, the PLP function Γ_{n_0} can have varying amplitude values $\Gamma_{n_0}(p_i)$ for peak positions $p_i \in P$, which directly fall at the decision line, such as the center time position of the PLP buffer ($p_i = n_0$). These amplitude values serve as indicators of **beat stability**, as depicted in Figure 3.2c. A high peak amplitude $\Gamma_{n_0}(p_i)$ signifies a **stable beat** situation, indicating that the neighboring optimal kernels κ_ℓ were similar in tempo and constructively added up over time (constructive interference). Conversely, a low peak amplitude indicates an **unstable beat** situation, where neighboring optimal kernels κ_ℓ with different tempi have canceled each other out (destructive interference). To this end, beat stability values can be utilized to control parameters in real-time applications, such as the volume of an accompaniment track, as detailed in Section 5.1.

Between the minimum and maximum peak amplitudes lies an entire **stability range**. Since the PLP function Γ_{n_0} is normalized (see constant C from Equation 3.7), the beat stability can have values between $[0 : 1]$. In this way, the peak amplitudes act as a **confidence metric** that can be used in addition to beat

detection. For example, if an application requires consistent beat output sequences without unstable and noisy beat sections, a **stability threshold** can be introduced. This threshold can serve as an optional beat filter that allows beats to be processed only when they surpass a certain amplitude value, as we will discuss in Section 5.2.

3.4.5 Inter Beat Interval and Local Tempo

Providing a local tempo measure for the current time position n_0 is particularly beneficial for interactive music making (Section 5.1), where the tempo can serve as an input parameter for time-based music instruments and effects plugins (e.g., sampler, reverb, delay, or echo). With PLP, we have two different approaches for determining local tempo. First, on the frame level, the PLP kernel κ_ℓ is calculated based on the tempo parameter τ'_n, which directly yields a local tempo value. Second, the PLP function Γ_{n_0} offers a more consistent tempo measure, defined by the inter beat interval of two consecutive peaks, as illustrated in Figure 3.2d.

The tempo output can be bounded by setting minimum and maximum tempo values for the PLP procedure, thereby defining a specific tempo range Θ. Modifying this tempo range allows us to influence the pulse level at which the beat tracker should operate, such as the normal tactus level (quarter notes) or one tempo octave higher with double tactus level (eighth notes). For instance, if the expected tempo for a beat tracking application is around 100 BPM, we can focus the tempo output on the range $\Theta = [80 : 120]$ for normal tempo or choose $\Theta = [180 : 220]$ to force double tempo output.

3.5 Experiments

To evaluate the described methods, we carry out multiple experiments. First, we compare our method with various low-latency beat trackers under specific oracle conditions and against methods from the existing literature for beat performance, latency, and tempo range. With the second experiment, we concentrate on an assessment of context-sensitive beat evaluation on different tempo ranges. Third, we evaluate our real-time procedure across different kernel sizes. Last, in the fourth experiment, we investigate the impact of the lookahead parameter on beat detection performance.

3.5.1 Datasets

For our experiments, we employ a diverse set of commonly used datasets. We report on the average track duration, tempo, and stability in Table 3.1. To calculate tempo stability, we convert all inter beat intervals to tempo values and normalize them by dividing each by its respective average track tempo, maintaining a tolerance interval of ±4% for stable tempi. For a formal definition of tempo stability, we refer to Schreiber et al. [118].

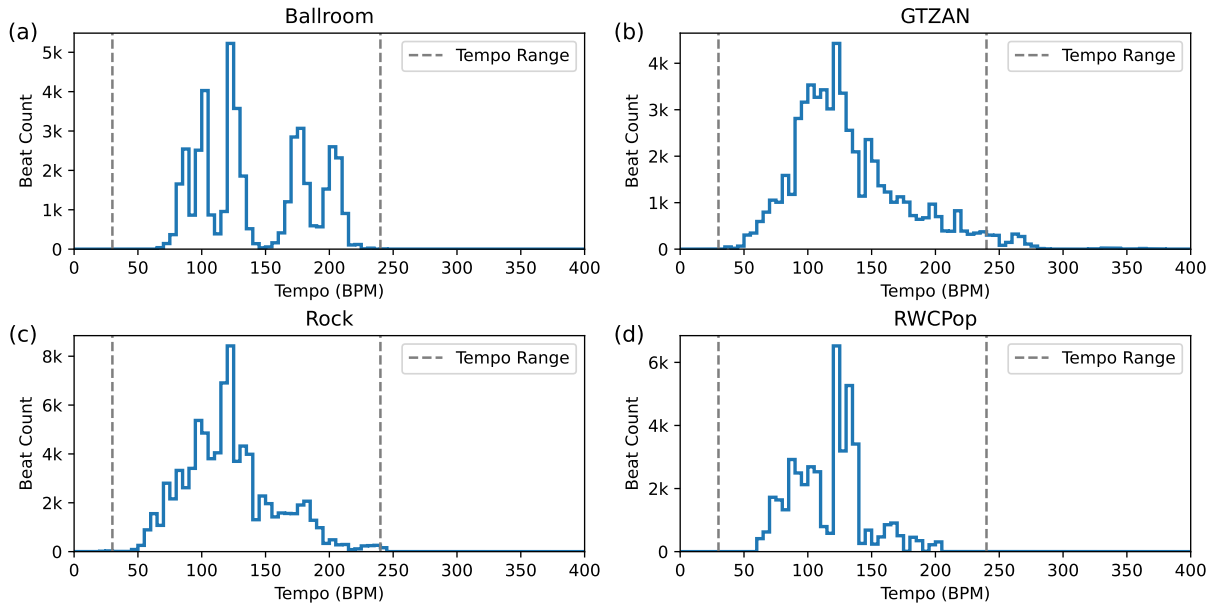


Figure 3.3: Beat-wise distribution of inter beat intervals (IBIs) in various datasets, considering a tempo resolution bin size of 5 BPM. The tempo range for our online model (30 – 240 BPM) is indicated with dashed lines.

Dataset	**Dataset** (Total)			**Track** (Average)		
Name	Tracks	Length	Type	Duration	Tempo	Stability
`Ballroom`	698	6h 03m	Excerpt	31± 1 s	129.7±39.7 BPM	89.0±13.4 %
`GTZAN`	993	8h 16m	Excerpt	30± 0 s	119.4±39.6 BPM	90.5±17.2 %
`Rock`	200	12h 53m	Full	232±91 s	115.7±34.7 BPM	81.6±15.0 %
`RWCPop`	100	6h 46m	Full	244±41 s	111.7±27.3 BPM	97.9±10.8 %

Table 3.1: Overview of the datasets used for evaluation.

The `Ballroom` dataset, introduced by Gouyon et al. [52], provides audio excerpts categorized by different dance music styles, displaying a wide variety of tempo ranges. `GTZAN`, described by Tzanetakis and Cook [132], comprises audio excerpts spanning a diverse range of genres, including Country, Metal, Hip-hop, Reggae, Jazz, and Classical. The `Rock` dataset, as introduced by Clercq and Temperley [26], features songs listed in Rolling Stone magazine's "500 Greatest Songs of All Time" and generally exhibits lower overall tempo stability compared to the other datasets listed. Additionally, the `RWCPop` dataset described by Goto et al. [51] provides a collection of pop songs with full audio recordings and high tempo stability.

Figure 3.3 offers an overview of the datasets, showing that `Rock` and particularly `GTZAN` include tempo values exceeding the range $\Theta = [30 : 240]$ handled by our online model, see also Subsection 3.5.5. Given that online models typically function within a more limited tempo spectrum compared to offline models, it is important to note that the wider tempo span of these datasets could potentially affect the overall performance of beat estimation.

3.5.2 Activation Functions

For the activation functions, we explored two distinct online methods. The first method (`RNN`) involves a machine learning approach, utilizing an online RNN activation with a specifically chosen single long short-term memory (LSTM) model [9]. In particular, we adopted the approach utilizing `madmom`'s `RNNBeatProcessor(online=True)` along with their pre-trained LSTM model 4 [11]. It is worth noting that these LSTM models are trained on the `Ballroom` dataset and therefore may exhibit superior performance on it compared to other datasets. The second method (`GT`) is a "ground truth" activation, which is derived from annotated beat positions. For each time position n, it outputs an amplitude value of 1 at annotated beat positions and 0 otherwise. To achieve this, we convert annotated beat positions (in seconds) into frame indices based on the frame period T (see Equation 3.10). This approach is valuable for analyzing post-processing methods independently of the activation, assuming best possible activation performance.

3.5.3 Post-Processing Methods

In this chapter, the term "post-processing" is used for a beat tracking method that determines beat positions from an activation function. Specifically, our PLP-based post-processor takes an activation function as input, generates the PLP pulses as described in Section 3.2, and detects the beats accordingly. For our experiments, we outline the following post-processing methods: As a baseline reference, labeled as `PLP-Off`, we use the original PLP concept to directly compare the transition from offline to online. Our online adaptation of the original PLP method is referred to as `PLP-On`.

3.5.4 Evaluation Measures

For evaluating beat estimation, we employ the standard F1-score metric with a tolerance window of ±70 ms, as implemented in `mir_eval` [107]. For context-sensitive evaluation, we apply the L-correct metric, as introduced by Grosche and Müller [53]. With this evaluation method, we consider not just a single beat but a series of consecutive estimated beats, each with a temporal context of length $L \in \mathbb{N}_{\geq 2}$. This is similar to how a listener would tap along to music, often needing a sequence of beats to adapt to tempo changes. Therefore, context-sensitive evaluation is valuable for evaluating how well our method can handle larger beat contexts (Subsection 3.4.2). To avoid initialization artifacts in the evaluation (see Subsection 2.2.2), we disregard all beats occurring before 5 s of each music track.

3.5.5 Experiment Setup

For our experiments, we focus on an application-ready setup, using an audio sampling rate $F_s = 44100$ Hz and a frame size $B = 512$ samples. These configurations are typical for real-time audio hardware, such as an audio interface, resulting in a frame period $T = 11.61$ ms. The default tempo range for our experiments

Model	Mode	Comments	F1-score (%)	Latency (ms)	Tempo (BPM)
RNN-PLP-On	Online	our model	74.72	11.61	30 - 240
RNN-PLP-On-Zero	Online	our model (zero latency)	74.68	**0.00**	30 - 240
Exploratory Studies: Oracle Conditions					
RNN-PLP-On-TR40	Online	(C1) use avg. track tempo	75.11	11.61	track (mean) ±40%
GT-PLP-On	Online	(C2) use GT activation	91.93	11.61	30 - 240
RNN-PLP-Off	Offline	(C3) use non-causal data	79.07	–	30 - 240
RNN-PLP-Off-TR40	Offline	(C4) use avg. track tempo	82.00	–	track (mean) ±40%
GT-PLP-Off	Offline	(C5) use GT activation	97.83	–	30 - 240
Methods Overview: Comparing with Literature					
BEAST-1	Online	[23]	80.04	46.44	55 - 215
Novel-1D	Online	[60]	76.48	20.00	55 - 215
BeatNet	Online	[59]	75.44	20.00	55 - 215
Böck-FF	Online	[10]	74.18	46.44	55 - 215
SpecTNT-TCN	Offline	[63]	88.7	–	–
Transformer	Offline	[139]	88.5	–	–
TCN	Offline	[8]	88.5	–	–

Table 3.2: Comparing various low-latency online beat trackers under specific conditions (C1, . . . , C5) and against existing literature for beat performance, latency, and tempo range, utilizing the GTZAN dataset. A tempo range of ±40% of the average track tempo is denoted by TR40 and ground truth activation by GT.

is $\Theta = [30 : 240]$, unless stated otherwise. With a tempo of 30 BPM, the beat events are 2 s apart, which can be considered as approaching the lower bound of human tempo perception. With an upper tempo of 240 BPM, we cover a full range of three tempo octaves (doubling the tempo of 30 BPM for three times). With TR40, we denote a tempo range of ±40% of the average track tempo, assuming that the mean tempo of a music recording is given. Throughout this chapter, we consistently use the PLP method with a fixed kernel size $K = 2N + 1$ of 6 s for all experiments and examples. Further discussion on this choice is provided in Subsection 3.6.3.

3.6 Results and Discussion

3.6.1 Overview and Methods Comparison

In Table 3.2, we compare various beat tracking methods for F1-score, latency, and tempo range to assess the performance of our models on the GTZAN dataset. The latency values listed are based on the individual frame period T of each respective method (see Equation 3.10). Our default online model RNN-PLP-On uses the RNN activation as input for the PLP-On post-processing and achieves an F1-score of 74.72%, which falls within the range of most other online beat trackers. However, our approach has only a very small latency of 11.61 ms. The leading online model in our list, BEAST-1, achieves an F1-score of 80.04%, but it comes with a significantly higher latency of 46.44 ms. This delay could be perceived as distinct acoustic events

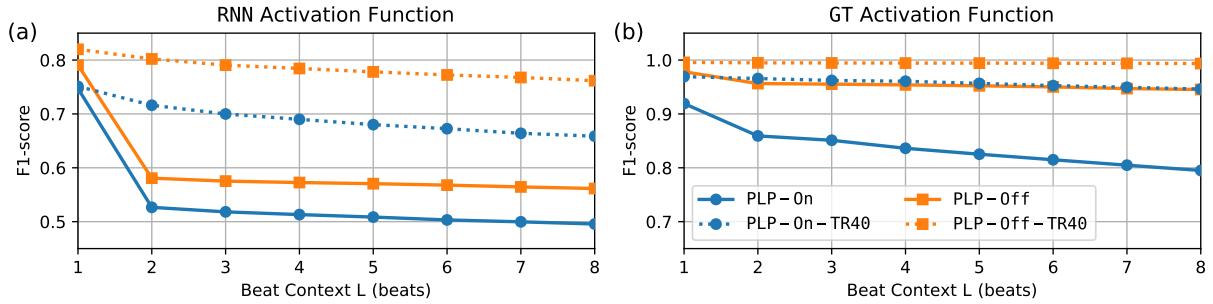


Figure 3.4: F1-score and L-correct metric for different activation functions and various post-processing methods on the GTZAN dataset. A tempo range of ±40% average track tempo is denoted by TR40.

by human ears, making it unsuitable for certain real-time audio applications where precise synchronization between input and output audio streams is important. Furthermore, we introduce RNN-PLP-On-Zero with an F1-score of 74.68%, which stands out as the only model in the table with a latency of 0 ms. Zero latency is achieved by offsetting the frame period $T = 11.61$ ms (see RNN-PLP-On) with a lookahead of 1 frame, allowing beats to be triggered 11.61 ms ahead of time. With RNN-PLP-On-TR40 and GT-PLP-On, we showcase potential enhancements of our online model when utilizing oracle conditions, such as a known average tempo range (TR40) or a perfect ground truth activation function (GT). In addition, we report on offline PLP models RNN-PLP-Off, providing a more comprehensive comparison between offline to online processing.

3.6.2 Context-Sensitive Evaluation on Tempo Range

With Figure 3.4 we report on results using the context-sensitive L-correct metric for different tempo ranges. When analyzing our online model PLP-On with the activation function RNN, we observe a decrease in F1-score from 74.72% (no context) to 52.66% ($L = 2$). This reflects the fact that detecting a series of beats accurately is more challenging than detecting a single beat. Further increasing the number of consecutive beats from $L = 2$ to $L = 8$, the F1-scores remain almost constant, with a slight decrease of about 3.09% (from 52.66% to 49.57%). This implies that the PLP method is inherently designed to effectively handle larger beat contexts, a characteristic that holds true for the online method as well.

When analyzing RNN-PLP-On-TR40, which utilizes a tempo range of ±40% of the average track tempo, we notice a much smaller decrease in F1-score from 75.11% (no context) to 71.61% ($L = 2$). Furthermore, for all other consecutive beats from $L = 2$ to $L = 8$, the F1-scores remain consistently higher compared to RNN-PLP-On, with only a slight decrease of approximately 5.74% (from 71.61% to 65.87%). Similarly, for the activation functions GT depicted in Figure 3.4b, we observe consistent F1-scores, particularly evident for the tempo range TR40. This suggests that a notable improvement in context-sensitive beat performance can be achieved if the average tempo of the analyzed music is known and utilized for specific beat tracking tasks, thus highlighting the tempo range as a valuable hyperparameter for controlling real-time applications.

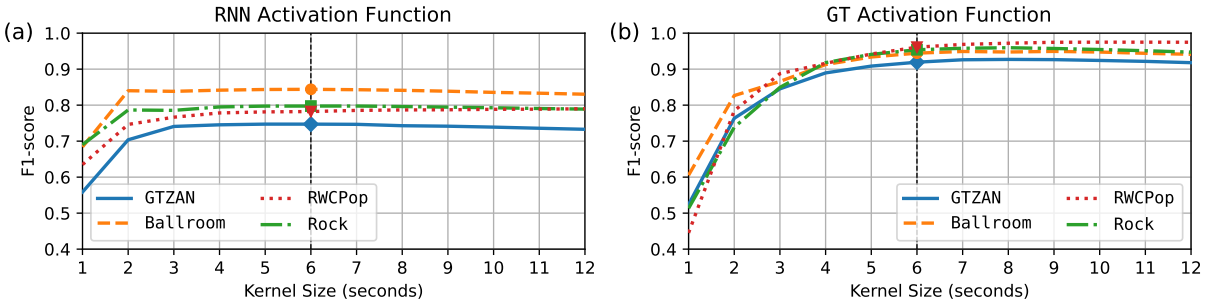


Figure 3.5: F1-score for different kernel sizes of PLP-On across various activation functions on different datasets.

3.6.3 Kernel Size Evaluation

In Figure 3.5, we assess our real-time procedure across various kernel sizes. Considering the best possible activation function GT in Figure 3.5b, we observe consistent behavior across all datasets. When the kernel size is above 6 s, the F1-score remains stable; however, when it falls below 6 s, the F1-score starts to drop significantly. Based on this observation, we opted to fix the kernel size at 6 s for our experiments, aiming to find a balance between stability in beat estimation and responsiveness to tempo changes in the music. A more realistic scenario is depicted with the activation function RNN in Figure 3.5a. In this case, we observe minimal variation in F1-score across different kernel sizes across various datasets. The F1-score begins to decrease only when the kernel sizes fall below 3 s. This suggests that our real-time procedure can effectively accommodate a wide range of kernel sizes, enabling adjustment of the beat context with minimal impact on F1-score.

3.6.4 Lookahead Impact Analysis

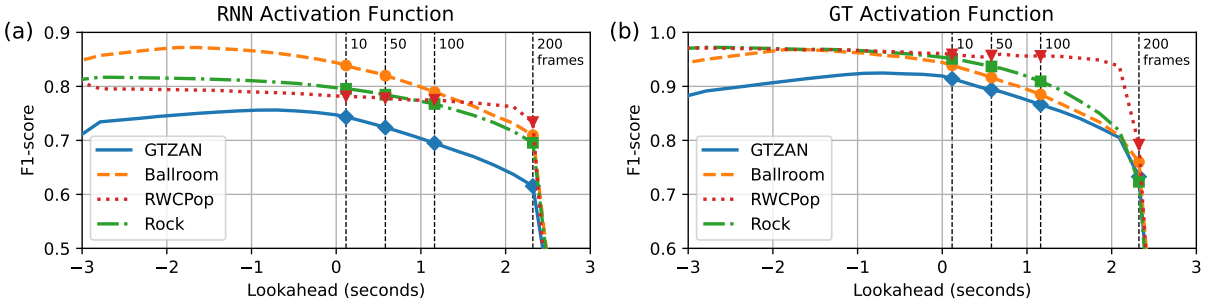


Figure 3.6: F1-score of various settings for lookahead of PLP-On for various activation functions across different datasets, see Table 3.3 for numbers.

We now discuss the impact of the lookahead parameter on the beat detection performance and start with Figure 3.6a, focusing on RNN and GTZAN as an example. The corresponding data is reported in Table 3.3. For a lookahead of 0 frames (0 ms), the F1-score is at 74.72%, which we already reported for RNN-PLP-On in Table 3.2. Using a lookahead of 1 frame (11.6 ms), there is only a small difference (−0.04%) in F1-score

Settings		F1-score (%) vs. Lookahead					
Lookahead in frames (ms)		0 (0.0)	1 (11.6)	10 (116.1)	50 (580.5)	100 (1161.0)	200 (2322.0)
RNN	GTZAN	74.72	74.68 (-0.04)	74.31 (-0.41)	72.39 (-2.33)	69.49 (-5.23)	61.54 (-13.18)
	Ballroom	84.39	84.34 (-0.05)	83.87 (-0.52)	81.98 (-2.41)	78.95 (-5.44)	70.94 (-13.45)
	RWCPop	78.22	78.21 (-0.01)	78.15 (-0.07)	77.80 (-0.42)	77.52 (-0.70)	73.38 (-4.84)
	Rock	79.74	79.72 (-0.02)	79.55 (-0.20)	78.45 (-1.29)	76.74 (-3.00)	69.55 (-10.19)
GT	GTZAN	91.93	91.87 (-0.05)	91.40 (-0.52)	89.39 (-2.54)	86.62 (-5.31)	73.27 (-18.65)
	Ballroom	94.44	94.39 (-0.06)	93.89 (-0.55)	91.63 (-2.81)	88.51 (-5.93)	75.98 (-18.46)
	RWCPop	96.10	96.08 (-0.02)	95.92 (-0.18)	95.71 (-0.39)	95.64 (-0.46)	79.20 (-16.90)
	Rock	95.39	95.36 (-0.03)	95.10 (-0.28)	93.70 (-1.69)	90.93 (-4.45)	72.33 (-23.06)

Table 3.3: The F1-score of lookahead settings in frames (and milliseconds) of PLP-On for different activation functions across different datasets, with each F1-score accompanied by the difference (in parenthesis) to the zero lookahead.

for a total of 74.68%. Using a 10-frame lookahead (116.1 ms) results in 74.31% (−0.41%), which is useful for compensating network and processing delays, as demonstrated in Section 5.1. With 50 frames (580.5 ms), comparable to the inter beat interval size (500 ms) at 120 BPM, the F1-score drops to 72.39% (−2.33%). At 100 frames (1161.0 ms), the F1-score falls to 69.49% (−5.23%), and at 200 frames (2322.0 ms), it declines significantly to 61.54% (−13.18%). Overall, the impact of lookahead on the F1-score is relatively small, especially for values below 50 frames (580.5 ms), emphasizing its importance as a valuable parameter for controlling and compensating latency in real-time beat tracking systems.

Comparing different datasets for lookahead performance, we observe a correlation with the tempo deviation of each dataset (see Table 3.1), which indicates the level of tempo variation across all songs. The dataset RWCPop exhibits the highest stability (97.9 ± 10.8%) and the lowest tempo deviation (±27.3 BPM) among the datasets analyzed, experiencing the smallest drop in F1-score (−4.84%) at 200 frames lookahead. In contrast, at 200 frames lookahead, Rock (±34.7 BPM), GTZAN (±39.6 BPM), and Ballroom (±39.7 BPM) experience drops in F1-score of −10.19%, −13.18%, and −13.45% respectively. This suggests that lookahead can be employed with minor impact on beat detection performance, particularly for highly stable music genres such as Pop. This is particularly evident in the case of GT, Figure 3.6b, where the RWCPop dataset demonstrates stable F1-scores for up to 2 s of lookahead, distinguishing itself from other datasets.

Note that all datasets in Figure 3.6 show higher F1-scores for negative lookahead (adding latency to make a more accurate decision by waiting for future data). However, since our focus is more on compensating for latency rather than adding latency for real-time applications, we do not discuss this fact any further.

3.7 Conclusions

In this chapter, we introduced a real-time beat tracking system designed to deliver zero latency and enhanced controllability for interactive music applications. In addition to beat detection, our model generates

valuable supplementary outputs, including beat context, beat stability, and inter beat interval analysis for local tempo estimation. Leveraging the beat lookahead technique, our method effectively compensates for latency by up to several hundred milliseconds in real-time audio systems. Furthermore, our model demonstrates enhanced controllability, allowing real-time applications to adjust latency compensation, pulse level, and beat context. This versatility proves particularly beneficial for real-time beat tracking tasks, where maintaining synchronization between the analyzed input audio stream and the produced output audio stream is crucial, thereby preventing any noticeable delay in audio perception. We validated and tested the capability of our model through a series of experiments. Additionally, in Chapter 5, we present two real-world scenario applications, focusing on interactive music-making (Section 5.1) and educational music gaming (Section 5.2). As a result, our model serves as a practical and lightweight tool for musicians to fine-tune their real-time audio setups and achieve the desired latency perception, while also opening up new creative controllability for real-time beat tracking applications.

For future research, our goals include conducting additional experiments, such as evaluating the lookahead feature of our model on smaller tolerance windows and investigating beat performance on more challenging datasets. Additionally, we aim to enhance our real-time pipeline by integrating newly developed beat activation models into our post-processing method as they become available in the future. Recognizing the practical utility of our system, we are committed to continuing the development of interactive applications and demonstrations.

In the following Chapter 4, we will present a method for generating pulse-based control signals for beat-synchronous audio effects. These signals are obtained in real time from the PLP system described in this chapter and will form the basis for a beat tracking audio plugin designed for both studio mixing and live performance applications.

4 Pulse-Based Control Signals for Beat-Synchronous Audio Effects

This chapter is based on [89]. The first author, Peter Meier, is the main contributor to this article. Together with Simon Schwär and his supervisor Meinard Müller, he developed the ideas, formalized the mathematical description of the real-time pulse parameters, and wrote the paper. Furthermore, Peter Meier implemented the real-time approach and created a demo visualization. Simon Schwär developed an audio plugin and created mixing examples for the case studies.

Building on the real-time extension of PLP estimation introduced in Chapter 3, this chapter presents a novel method for generating diverse real-time control signals from the PLP output. While the PLP activation function encodes both predominant pulse information and pulse stability, we propose several normalization procedures to discern local pulse oscillation from stability, utilizing the PLP activation envelope. Through this, we generate pulse-synchronous low frequency oscillators (LFOs) and supplementary confidence-based control signals, enabling dynamic control over audio effect parameters in real time. Additionally, our approach enables beat position prediction, providing a lookahead capability, for example, to compensate for system latency. To showcase the effectiveness of our control signals, we introduce an audio plugin prototype designed for integration within a digital audio workstation (DAW), facilitating real-time applications of beat-synchronous effects during live mixing and performances (see Section 5.3). Moreover, this plugin serves as an educational tool, providing insights into PLP principles and the tempo structure of analyzed music signals. The remainder of this chapter is organized as follows. We start in Section 4.1 by introducing the topic and outlining our approach. Section 4.2 revisits the real-time PLP method and its central element, the PLP buffer, introducing and summarizing essential notation. Section 4.3 presents normalization techniques for disentangling beat structure from estimation confidence. Section 4.4 introduces four derived control signals for DAW parameter control, including low-frequency oscillators and beat confidences. Section 4.5 concludes this chapter. The following Chapter 5 demonstrates applications of the four control signal variants (see Section 5.3). Additional materials and audio examples are available online.[13]

[13] https://audiolabs-erlangen.de/resources/MIR/2024-DAFx-RealTimePLP

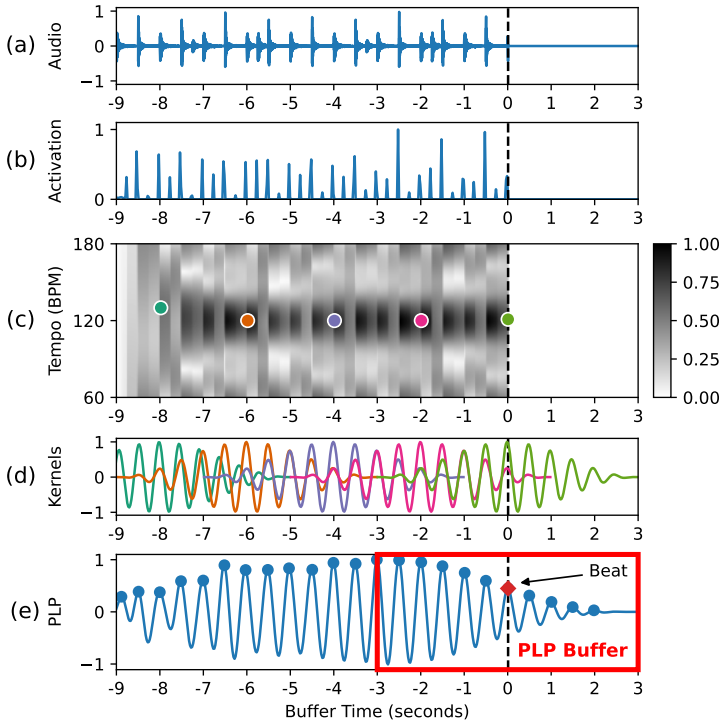


Figure 4.1: The real-time PLP procedure is showcased utilizing a straightforward drum beat as follows: **(a)** Audio signal. **(b)** Activation function. **(c)** Tempogram. **(d)** Pulse kernels. **(e)** PLP function with buffer.

4.1 Introduction

Many audio effects and instruments, such as echo, flanger, tremolo, or synthesizers, are often strongly aligned with the rhythmic structure of the music [141]. This is typically achieved by modulating effect parameters like amplitude, frequency, or phase in a beat-synchronized way with a low frequency oscillator [140]. LFOs are used to generate various waveforms such as square, sawtooth, or sinusoidal and typically operate at frequencies below 10 Hertz, so that they can enable rhythmic variations of parameters.

The challenge with LFO-modulated and beat-synchronous effects lies in the manual tuning of LFO frequency and phase to be in sync with the rhythmic structure of the music. In DAWs, users often find themselves in need of manually adjusting parameters or drawing automation curves to achieve precise synchronization. This process can be time-consuming and requires careful attention to detail. Similarly, in live performance settings such as using a guitar effects pedal, musicians must manually enter the desired tempo through tapping, which can be cumbersome and prone to human error, while there is also no way to dynamically react to natural tempo variations. As a result, automating these processes can greatly enhance efficiency and accuracy in applying beat-related effects.

Drawing from the predominant local pulse (PLP) concept [53], which was originally developed for offline applications, we introduced a real-time PLP tracking variant in the previous Chapter 3. This approach involves transforming the audio signal into an activation function, computing a Fourier tempogram to identify local periodic patterns, as well as selecting and overlap-adding windowed sinusoidal kernels that represent the local pulse structure, as outlined in Figure 4.1. The PLP buffer, which is updated with each

new frame of audio analysis, serves as a central real-time component, reflecting both pulse oscillation and beat stability.

As the main contribution of this chapter, we introduce a method to transform and normalize the real-time PLP output to derive multiple control signals. Given that PLP is based on oscillation, with sinusoidal kernels representing the local pulse structure of the audio input, the PLP output closely resembles an LFO signal. We introduce various normalization steps aimed at separating pulse oscillation from stability measures. Specifically, we develop two different confidence envelopes and two different oscillation signals: One for a beat-synchronized LFO signal with confidence amplitude, and another for a beat-synchronized LFO signal with constant amplitude, achieved by normalizing with the confidence envelope. In the following Chapter 5, we will explore several case studies illustrating how the derived LFO and confidence parameters facilitate real-time applications of beat-synchronous effects in live performances and mixing environments (see Section 5.3). Through these case studies, we demonstrate how the derived control signals not only enable creative sound design but also provide valuable insights into the tempo structure of the audio recording.

4.2 Predominant Local Pulse

In this section, we introduce the main idea of the PLP algorithm as first introduced in [53]. Specifically, we summarize the real-time PLP procedure, as initially described in [88], and discuss the main properties of the PLP buffer. This buffer serves as the centerpiece of the real-time PLP procedure, from which we aim to extract control signals.

4.2.1 Real-Time PLP Procedure

This section summarizes the mathematical notation required for this chapter. For a more detailed description, we refer to [53] and [88]. With Figure 4.1, we illustrate the basic idea of the real-time PLP procedure. First, the audio signal (Figure 4.1a) is transformed into an activation function $\Delta : \mathbb{Z} \to \mathbb{R}$, analyzing spectral changes over time positions $n \in \mathbb{Z}$. Note that the term **activation function** can refer to both novelty functions (such as spectral flux) and probabilistic machine learning models (such as an RNN). To detect local periodic patterns in Δ (Figure 4.1b), we calculate a Fourier tempogram $\mathcal{T}(n, \tau)$, which is a function over time positions n and tempo parameters τ, as depicted in Figure 4.1c (for details we refer to [53]). For this purpose, we use a window[14] function $\mathcal{W} : [-N : N] \to \mathbb{R}$ for $N \in \mathbb{N}$, which is normalized and centered around each time position n, yielding a total window size of $L = 2N + 1$. For any arbitrary but fixed time position $n \in \mathbb{Z}$, the window $\mathcal{W}$ specifies a neighborhood indexed by $m \in [n - N : n + N]$.

[14] We use a Hann window for all the illustrations in this chapter, as well as for our audio plugin prototype.

For each $n \in \mathbb{Z}$, we select a windowed sinusoidal kernel $\kappa_n : [n - N : n + N] \to \mathbb{R}$, given by

$$\kappa_n(m) := \mathcal{W}(m - n) \cdot \cos\Big(2\pi\big((\tau_n/60) \cdot m - \varphi_n\big)\Big), \tag{4.1}$$

for $m \in [n-N : n+N]$, that best aligns with the periodic structure of the activation function Δ. The tempo parameter τ_n and the corresponding phase parameter φ_n maximize the tempogram $\mathcal{T}(n, \tau)$, as illustrated by colored dots in Figure 4.1c and can be obtained from the complex-valued Fourier representation underlying the tempogram [88]. In a real-time context, we consider n_0 as the current time position, where we only have access to the beat activation values $\Delta(n)$ for all time positions $n \leq n_0$. For the current time position n_0, we obtain a real-time PLP function $\Gamma_{n_0} : [-\infty : n_0 + N] \to \mathbb{R}$ with

$$\Gamma_{n_0}(n) := \frac{1}{C} \sum_{\ell=n-N}^{n_0} \kappa_\ell(n), \tag{4.2}$$

which is defined for all time positions $n \in [-\infty : n_0+N]$ and has access to all kernels κ_ℓ for $\ell \in [n-N : n_0]$. The constant

$$C = \sum_{n=-N}^{N} \mathcal{W}(n) \tag{4.3}$$

ensures that the values of $\Gamma_{n_0}(n)$ lie within a range of $[-1 : 1]$. The PLP buffer, depicted in Figure 4.1e, displays only the section of $\Gamma_{n_0}(n)$ for $n \in [n_0 - N : n_0 + N]$, containing all the necessary information to compute the buffer for the subsequent time position.

4.2.2 Extracting Control Signals from PLP Buffer

The PLP buffer, as introduced in Subsection 4.2.1, serves as the central component of the real-time PLP procedure and is updated with the audio data for each new current time position n_0. In addition to encoding the local pulse structure as oscillations, it also exhibits varying amplitudes, reflecting the local tempo stability of the audio signal, see [53]. The reason for the varying amplitudes is that, depending on the local tempo structure and the predominant tempo kernels selected, different kinds of interference between kernels can occur, as illustrated in Figure 4.2.

When the tempo situation is stable, as depicted in Figure 4.2a, the kernels κ_ℓ selected from the tempogram $\mathcal{T}$ have a similar frequency and overlap-add constructively, leading to a PLP function Γ_{n_0} with high amplitude values. Conversely, when the tempo situation is unstable, as depicted in Figure 4.2b, the neighboring kernels exhibit significant frequency variations and cancel each other out in the overlapping section, resulting in a lower overall amplitude of the PLP function. In this way, the PLP buffer contains not only pulse oscillation but also a beat stability measure in a single representation.

One inherent feature observed in the PLP buffer is that the values of its right half fade out to zero. This occurs because there are no kernels κ_ℓ available for overlap-adding at future time positions $n > n_0$,

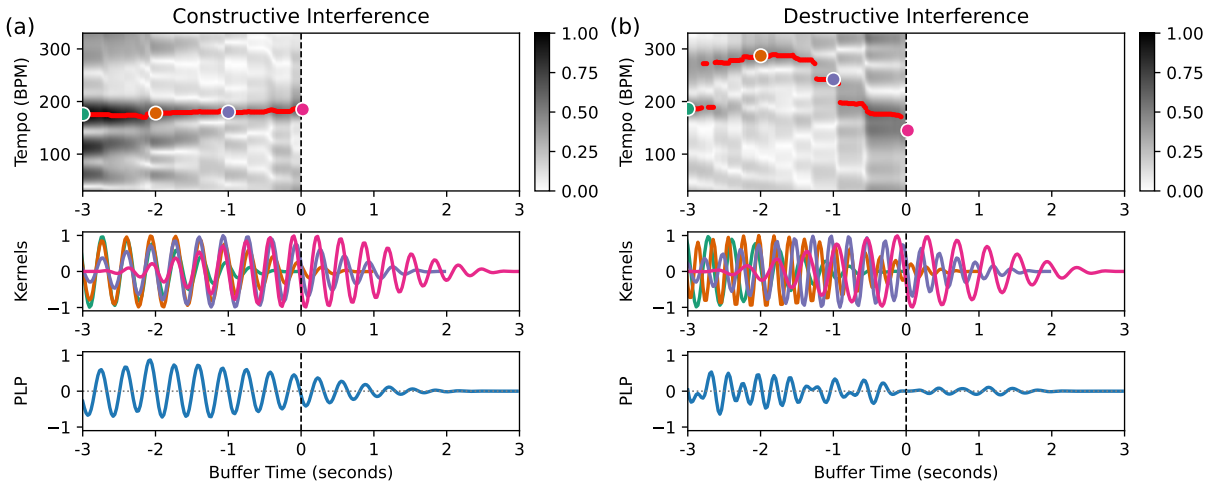


Figure 4.2: The tempogram, kernels, and PLP buffer illustrate **(a)** constructive interference and **(b)** destructive interference.

corresponding to diminishing confidence in pulse information the further one looks into the future. While this aspect is not crucial for beat detection, as it has been successfully employed in other real-time applications before [84, 85], normalizing the PLP buffer is especially necessary for extracting control signals to be independent from beat confidence.

In the following, we address two challenges for extracting control signals that arise from the varying amplitudes in the PLP buffer. First, to achieve a more consistent oscillation, compensation for the fading amplitude within the PLP buffer should be enforced. Second, the pulse information should be separated into pulse oscillation and beat stability measure. In Section 4.3, we will examine the normalization process to bring the PLP buffer into a more standardized form.

4.3 Normalization

To address the challenges outlined in Subsection 4.2.2, in this section, we explore various layers of normalization for the PLP buffer. First, in Subsection 4.3.1, we introduce a global kernel window function, followed by a PLP envelope function in Subsection 4.3.2. Subsequently, we utilize these functions to normalize the PLP buffer in multiple steps, as described in Subsection 4.3.3.

4.3.1 Overlap-Add of Kernel Window Functions

PLP is computed by adding overlapping kernels κ_ℓ, each obtained by multiplying a window function $\mathcal{W}$ with a maximum amplitude of one. To determine the maximum possible amplitude of the PLP function Γ_{n_0}, we assume perfect constructive interference of the shifted kernels and compute the overlap-add of the kernel window functions $\alpha_{n_0} : [n_0 - N : n_0 + N] \to \mathbb{R}$, defined by

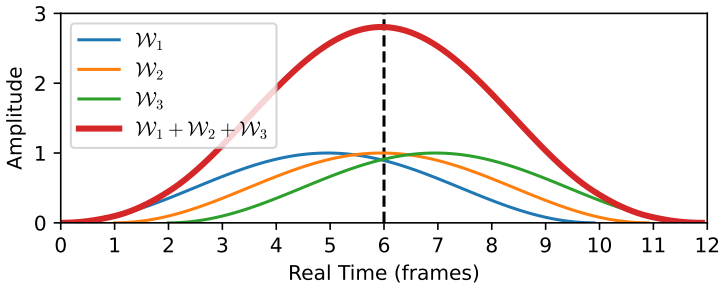


Figure 4.3: An example demonstrating three shifted PLP kernel windows (Hann function) contributing to an overlap-added kernel window function.

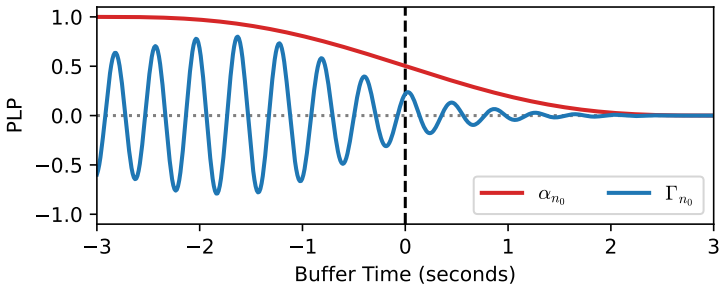


Figure 4.4: The PLP function Γ_{n_0} with the overlap-added kernel window function α_{n_0} indicating maximum boundaries.

$$\alpha_{n_0}(n) := \frac{1}{C} \sum_{\ell=n-N}^{n_0} \mathcal{W}(n-\ell), \tag{4.4}$$

for $n \in [n_0 - N : n_0 + N]$. The constant C, as defined in Equation 4.3, ensures that α_{n_0} lies within the range of values $[0 : 1]$. Since α_{n_0} from Equation 4.4 is shift invariant and does not change for different values of n_0, we can pre-calculate this normalization function based on the window configuration alone.

For demonstration purposes, in Figure 4.3, we illustrate only three shifted and overlap-added kernel windows $\mathcal{W}_1, \mathcal{W}_2$, and $\mathcal{W}_3$ that combine to form a sum of kernel windows with an overall higher amplitude. The exact maximum value of that sum varies depending on factors such as the hop size, window length, window type, and the total number of overlapping kernels. When running a real-time PLP procedure, all of these settings are known and can be pre-computed to remain constant during runtime.

In this way, α_{n_0} provides an important stage of the PLP buffer normalization, as depicted in Figure 4.4. The overlap-add of kernel window functions α_{n_0} acts as the upper limit of the PLP function Γ_{n_0} across all time positions n and also dictates the shape of the fading amplitude within the PLP buffer for $n \in [n_0 - N : n_0 + N]$. The closer the PLP function Γ_{n_0} approaches the boundaries of α_{n_0}, the more stable the local tempo structure of the audio is. To utilize the PLP buffer without the influence of the fading amplitude, we can define an α-normalized variant of Γ_{n_0}, denoted by

$$\Gamma^{\alpha}_{n_0} := \frac{\Gamma_{n_0}}{\alpha_{n_0}}. \tag{4.5}$$

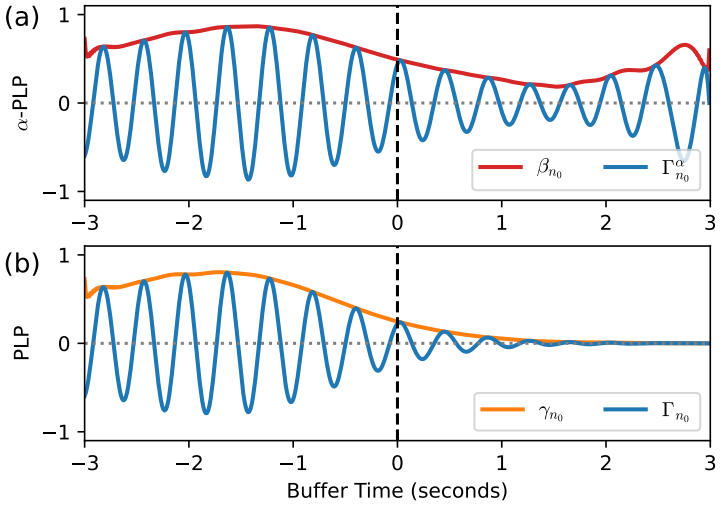


Figure 4.5: (a) The α-normalized PLP function $\Gamma^{\alpha}_{n_0}$ with envelope β_{n_0}. **(b)** The PLP function Γ_{n_0} with envelope γ_{n_0}.

4.3.2 PLP Buffer Envelope

Another measure of stability is derived from the PLP function itself, or more precisely, from the **envelope** formed by its peak positions, as illustrated in Figure 4.5a. The envelope β_{n_0} of the PLP function $\Gamma^{\alpha}_{n_0}$ represents the magnitude of its oscillation, which can be computed using the Hilbert transform [123]. The higher the value of β_{n_0}, the more stable the beat structure of the audio, and vice versa. In this way, the envelope β_{n_0} serves as a valuable analytical tool for expressing the beat stability, particularly when the PLP function Γ_{n_0} is already α-normalized. Note that, even though $\Gamma^{\alpha}_{n_0}$ falls within the value range $[-1:1]$, this does not automatically guarantee that the envelope β_{n_0} is limited to the range $[0:1]$. To ensure the desired normalization, in practice, we simply clip all values to 1 where $\beta_{n_0} > 1$.

Furthermore, the combination of α- and β-normalizations can also prove to be beneficial. For instance, when using future buffer time positions $n > n_0$, a descending slope fading to zero could be advantageous to express the uncertainty of the predictions. For this use case we define a second envelope

$$\gamma_{n_0} := \alpha_{n_0} \cdot \beta_{n_0}, \tag{4.6}$$

as illustrated in Figure 4.5b. While theoretically γ_{n_0} could be computed directly from the Hilbert transform of Γ_{n_0}, we found that cleaner signals are obtained when applying α-normalization first to prevent values from fading to zero. This approach can reduce unwanted edge effects when using the Hilbert transform.

4.3.3 PLP Buffer Normalization

With the overlap-added kernel window function α_{n_0} (Subsection 4.3.1) and the envelopes β_{n_0} and γ_{n_0} (Subsection 4.3.2), we have multiple layers for normalizing the PLP buffer Γ_{n_0}, as illustrated in Figure 4.6.

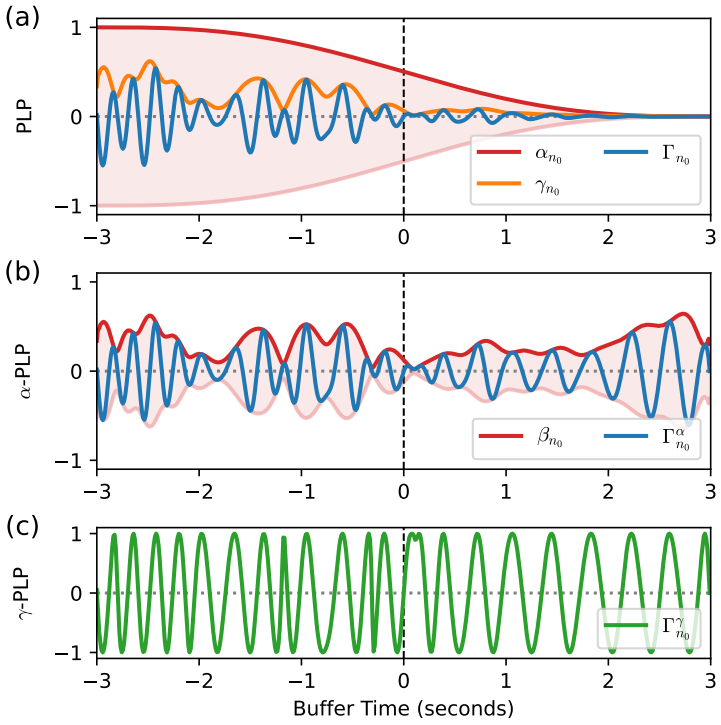


Figure 4.6: The PLP buffer normalization in three steps: **(a)** PLP function Γ_{n_0} with normalization α_{n_0} and envelope γ_{n_0}. **(b)** α-normalized PLP function $\Gamma^{\alpha}_{n_0}$ with envelope β_{n_0}. **(c)** γ-normalized PLP function $\Gamma^{\gamma}_{n_0}$.

Starting with Figure 4.6a, both Γ_{n_0} and α_{n_0} already fall within the range of $[-1:1]$ and $[0:1]$, respectively, using the constant C from Equation 4.3. Γ_{n_0} exhibits a fading amplitude within the PLP buffer, descending to values near zero, which can be compensated by normalizing with α_{n_0}. The resulting α-normalized PLP function $\Gamma^{\alpha}_{n_0}$ is illustrated in Figure 4.6b. Note how the normalization process of the α-PLP is especially relevant for the right side of the PLP buffer, where oscillations for future time positions $n > n_0$ are now clearly visible instead of fading to zero. In this state, we can consider $\Gamma^{\alpha}_{n_0}$ as a pulse oscillation modulated in amplitude by its beat stability, which is depicted by the envelope β_{n_0} in Figure 4.6b. Combining α_{n_0} and β_{n_0} results in an envelope γ_{n_0} that matches Γ_{n_0}, as illustrated in Figure 4.6a. To separate pulse oscillation from beat stability measures, we can either normalize $\Gamma^{\alpha}_{n_0}$ with its envelope β_{n_0} or directly normalize Γ_{n_0} with its envelope γ_{n_0}, which yields identical outputs. The resulting γ-normalized variant of the PLP function, denoted as $\Gamma^{\gamma}_{n_0}$, exhibits a pulse oscillation between $[-1:1]$ with constant amplitude but without the influence of beat stability, as depicted in Figure 4.6c.

With α_{n_0}, β_{n_0}, and γ_{n_0}, we now have multiple separate PLP normalizations that offer insights into the local pulse structure in different ways. In Figure 4.7, we provide two examples of how these normalizations behave for stable and unstable beat tracking scenarios. A stable beat tracking scenario, as depicted in Figure 4.7a, is characterized by consistently high values for the envelope β_{n_0}, with only small variations close to one. As a result, for $\Gamma^{\gamma}_{n_0}$ in Figure 4.7b, we observe a highly consistent pulse curve resembling a clean sinusoidal waveform with a regular frequency oscillation. An unstable beat tracking scenario, as illustrated in Figure 4.7c, results in significant fluctuation in the envelope β_{n_0}, with peak values close to

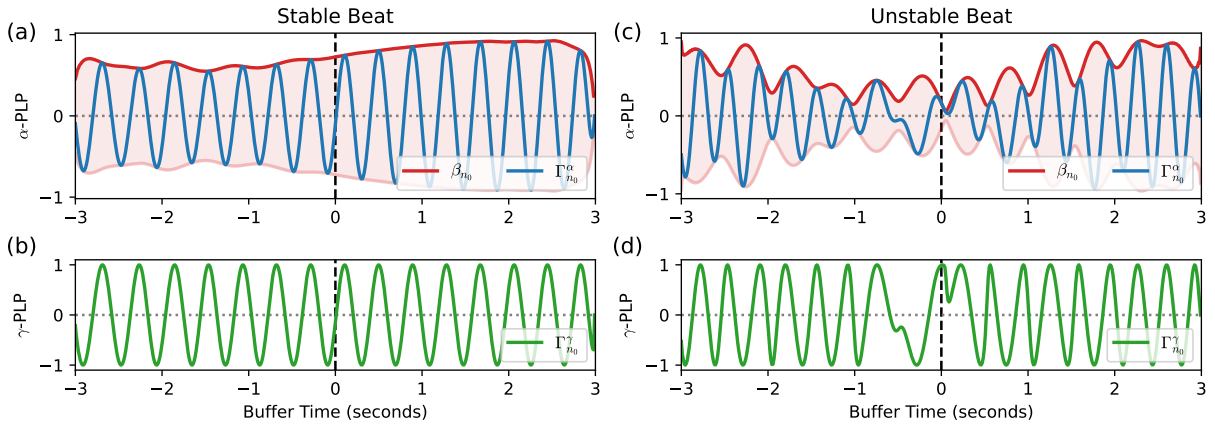


Figure 4.7: Illustration of stable and unstable beat scenarios for α-normalized and γ-normalized PLP functions.

zero. As a result, for $\Gamma^{\gamma}_{n_0}$ in Figure 4.7d, we observe a more inconsistent pulse curve with phase shifts indicating tempo changes and a higher amount of non-regular frequency modulations.

With this transformation of the PLP buffer into normalized versions $\Gamma^{\alpha}_{n_0}$ and $\Gamma^{\gamma}_{n_0}$, we can derive a variety of different control signals, as we will discuss in Section 4.4.

4.4 Control Signals

In Figure 4.8, we present an example of a real-time PLP procedure using a short audio excerpt. An animation of this audio excerpt is available on our supplemental website.[15] For Figure 4.8a, we display the waveform of the audio with a black cursor fixed at the current time position n_0. At this particular time position, we observe the internal state of the PLP buffer, with the current values for tempogram $\mathcal{T}(n,\tau)$, kernels $\kappa_\ell(n)$, and PLP function $\Gamma_{n_0}(n)$. Note that we distinguish between time axis for **real time**, such as the audio waveform, and **buffer time**, which serves as an internal time measure within the PLP buffer, referencing the neighborhood of the current time position n_0. For each time position n_0, we apply the normalization methods α_{n_0}, β_{n_0}, and γ_{n_0}, as described in Section 4.3. From these normalizations, we can derive distinct control signals, as discussed in the next subsections.

4.4.1 Beat Confidence

The α-normalized PLP contributes a control signal by utilizing the beat stability of the PLP function $\Gamma^{\alpha}_{n_0}$. Specifically, we define **beat confidence** at time position n_0 as the value of the envelope β_{n_0} at the central buffer position. The resulting control signal, labeled β-confidence, is depicted in Figure 4.8b.

[15] https://www.audiolabs-erlangen.de/resources/MIR/2024-DAFx-RealTimePLP

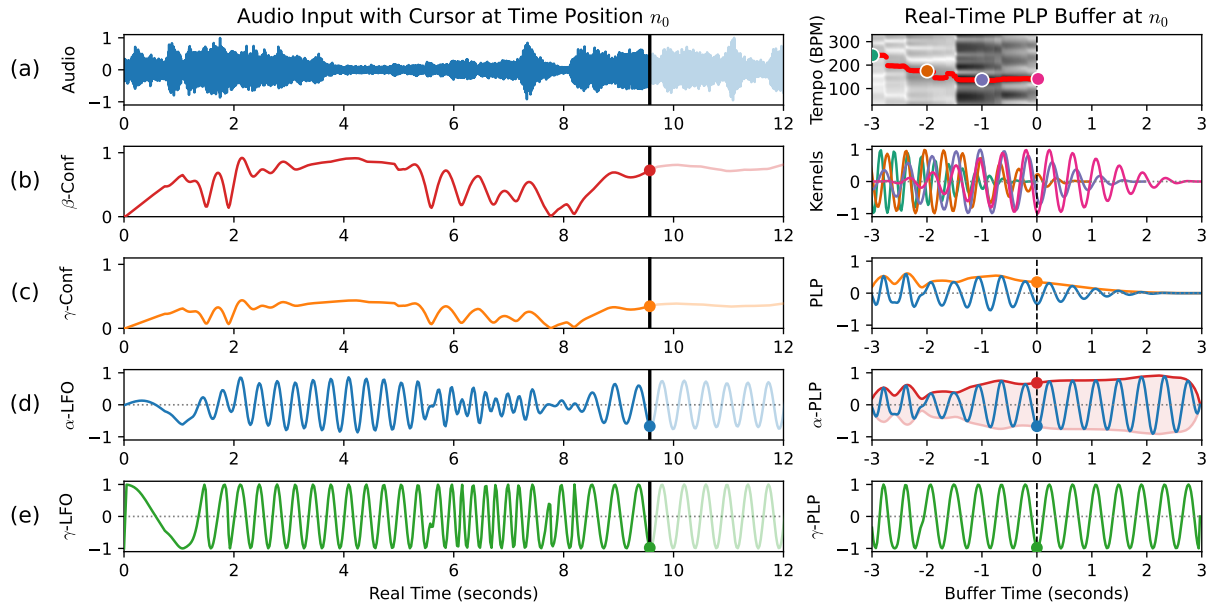


Figure 4.8: An audio excerpt featuring: **(a)** Audio input and control signals, **(b)** β-Conf, **(c)** γ-Conf, **(d)** α-LFO, and **(e)** γ-LFO. The black cursor indicates the current time position n_0. The right side of the figure displays the inner state of the PLP buffer at that time position.

In parallel, we can derive a similar control signal γ-confidence, which is based on Γ_{n_0} and captures the value of the envelope γ_{n_0} for each real time n_0, as depicted in Figure 4.8c. The γ-confidence shares similarities with the β-confidence but exhibits a lower amplitude. This characteristic could prove especially beneficial if we opt to use the PLP lookahead capability and adjust the buffer read position to future or past time positions, as we will discuss in Subsection 4.4.3. In such scenarios, the γ-confidence could effectively represent the integration of more uncertain future pulse information, encoded with a reduced amplitude.

To clarify the significance of beat confidence, let's examine the audio excerpt in Figure 4.8a. The lowest confidence value in the presented audio occurs around eight seconds of real time, where the music is fading out and no beat information is available. The subsequent section, spanning from seconds 8 to 12, is marked by consistent and rhythmic playing, resulting in a notably higher beat confidence.

4.4.2 Low Frequency Oscillator

The α-normalized PLP contributes another control signal by utilizing the pulse oscillation of the PLP function $\Gamma^{\alpha}_{n_0}$. Specifically, we define the **low frequency oscillator** α-LFO as the value of $\Gamma^{\alpha}_{n_0}$ at the central buffer position for each real time n_0. The corresponding control signal is shown in Figure 4.8d. Additionally, from $\Gamma^{\gamma}_{n_0}$ we can directly derive γ-LFO, which is depicted in Figure 4.8e. While the γ-LFO exhibits a clean, uniform oscillation between values of $[-1:1]$, the α-LFO combines both γ-LFO and β-confidence, with the confidence modulated onto the signal's amplitude.

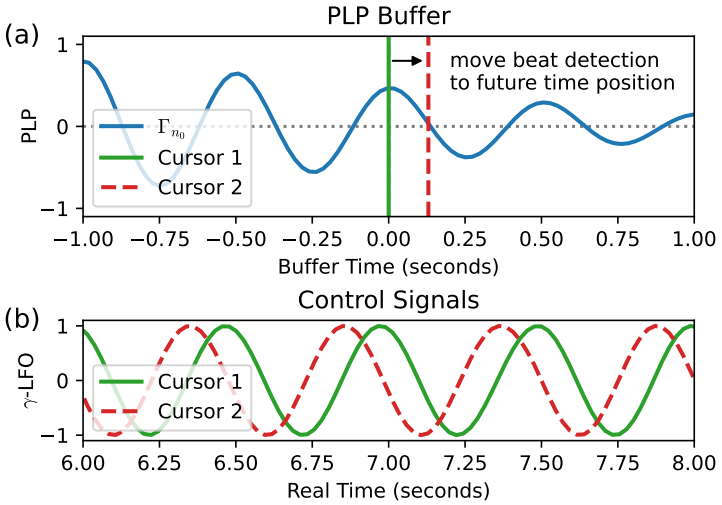


Figure 4.9: Demonstration of the PLP buffer with lookahead capability, shown with two different buffer read positions (cursors) for the γ-LFO control signal.

To explain the different behaviors of the LFOs defined, we can again utilize the example in Figure 4.8a. Both the α-LFO and the γ-LFO clearly visualize the tempo structure of the provided song. At the beginning of the audio excerpt, we observe a mid-tempo oscillation, followed by a sudden change to a high-tempo section around 5-8 seconds, and finishing with a low-tempo section towards the end of the audio track.

4.4.3 Lookahead Capability

For each time position n_0, we can analyze a PLP buffer where the left half of the buffer shows past pulse information and the right half predicts future pulse information. This prediction is based on the centered pulse kernels that drive the real-time PLP procedure (see Subsection 4.2.1). The central buffer position corresponds to the current time position and is typically employed to retrieve values for control signals, as described in Subsection 4.4.1 and Subsection 4.4.2. However, for the buffer read position, any buffer time can be chosen, leading to some useful and interesting time-related effects. For example, by shifting the buffer read position to future time positions, we can trigger beats earlier and obtain control signals that operate ahead of time, as illustrated in Figure 4.9.

For Figure 4.9a, we see a PLP function Γ_{n_0} and two different buffer read positions labeled as `Cursor 1` and `Cursor 2`. While `Cursor 1` is positioned at the central buffer position, `Cursor 2` utilizes the lookahead capability of real-time PLP and is shifted to a future time position. As a consequence, in Figure 4.9b, the γ-LFO for `Cursor 2` is consistently ahead of time in oscillation and reaches its peaks earlier than the `Cursor 1` version of the LFO. This can be advantageous for two distinct reasons. First, for technical purposes, it allows compensating for system latency effects and synchronizing the generated system output with the analyzed audio input even if the processing introduces an inherent delay. Second, for creative purposes, it enables time-related effects that need to start earlier to finish at the upcoming beat position, as detailed in Subsection 5.3.2. In this way, our system offers multiple adjustable control signals suitable for various applications, as we will discuss in Section 5.3.

4.5 Conclusions

In this chapter, we introduced a novel method for deriving pulse-synchronous LFOs and supplementary confidence-based control signals from a real-time PLP buffer. In this context, we discussed several normalization steps for the PLP output aimed at separating pulse oscillation from beat stability measures. With this, our goal is to provide a practical method for musicians and mixing engineers, allowing them to automate beat-synchronous audio effects in live performances or to creatively utilize real-time control signals for mixing applications. In the following Chapter 5, we will introduce several interactive beat tracking applications, including a pulse tracking audio plugin prototype (see Section 5.3), demonstrating creative applications of our method, including its lookahead capability, and offering educational insights into the real-time PLP procedure.

5 Interactive Beat Tracking Applications

In this chapter, we introduce three interactive beat tracking application prototypes that creatively leverage our real-time approach for predominant local pulse estimation, as detailed in Chapter 3, along with the pulse-based control signals described in Chapter 4. These applications explore a variety of beat tracking topics, including interactive music making (Section 5.1), educational music gaming (Section 5.2), and creative music production (Section 5.3).

In addition to the applications presented in the following sections, the beat tracking methods introduced in this part also formed the basis for two Bachelor theses, both closely supervised by Peter Meier. The first thesis, *Real-Time Beat Tracking for Creative Music Production*[16] [96], explored the integration of real-time beat tracking algorithms into DAWs to support creative workflows for music producers. The second thesis, *Automated Real-Time Beat Tracking: Response Time and Confidence Analysis*[17] [109], provided a comparative analysis of human and machine real-time beat tracking performance.

5.1 Interactive Music Making

> This section is based on [84]. The first author, Peter Meier, is the main contributor to this article. In collaboration with his supervisor Meinard Müller, he developed the ideas, designed the application, and wrote the paper. In addition, Peter Meier implemented all approaches and created the demo application system. Gerhard Krump was the supervisor of the main author at the Technische Hochschule Deggendorf.

We begin Chapter 5 with a demonstration on interactive music making. The terminal application, called "Beat Command Line Interface" (`beatcli.py`), is developed in Python and is based on the real-time procedure described in Section 3.3. In the following subsections, we first describe the command line interface in detail (Subsection 5.1.1), and then present a real-time accompaniment system as a practical use case (Subsection 5.1.2).

[16] `https://www.audiolabs-erlangen.de/resources/MIR/2024-RealTimeBeat-MusicProduction`
[17] `https://www.audiolabs-erlangen.de/resources/MIR/2024-ARTBeaT`

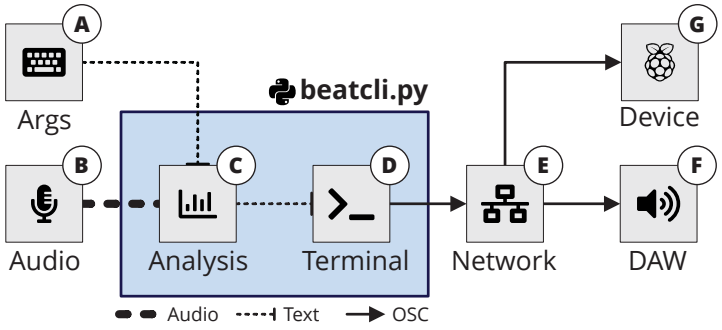


Figure 5.1: A block diagram of the beatcli.py terminal application. (A) Input arguments. (B) Audio input. (C) Audio analysis. (D) Terminal output. (E) Network output. (F) Receiving software. (G) Receiving hardware.

5.1.1 Beat Command Line Interface

With Figure 5.1, we show a block diagram of the application. It accepts input arguments (A) to select a device/channel of the audio input (B) and configure various parameters required for the audio analysis (C). It continuously executes a complete PLP real-time procedure and uses data buffering (Subsection 3.3.4) to update the PLP model for every new frame of audio provided by the input audio streaming (Subsection 3.3.2). Upon beat detection (Subsection 3.4.1) within the current time frame, both terminal output (D) and network output (E) with corresponding local pulse information are provided, as shown in Figure 5.3. This system output (Section 3.4) can be received by software clients (F), such as a DAW, or hardware devices (G), such as microcontrollers, to utilize the transmitted local pulse information.

Figure 5.2: The help function of the beatcli.py application with information about input arguments.

```
(.myenv) ➜  applications git:(develop) python beatcli.py -h
usage: beatcli.py [-h] [--device ID] [--channel NUMBER]
                  [--samplerate FS] [--blocksize SAMPLES]
                  [--tempo LOW HIGH] [--lookahead FRAMES]
                  [--kernel SIZE] [--ip IP] [--port PORT]

Beat (C)ommand (L)ine (I)nterface.

optional arguments:
  -h, --help            show this help message and exit
  --device ID           (6) device id for sounddevice input
  --channel NUMBER      (10) channel number for sounddevice input
  --samplerate FS       (44100) samplerate for sounddevice
  --blocksize SAMPLES   (512) blocksize for sounddevice
  --tempo LOW HIGH      ([60, 180]) tempo range in BPM
  --lookahead FRAMES    (0) number of frames (samplerate / blocksize)
                        to lookahead in time and get the next beat
                        earlier to compensate for latency
  --kernel SIZE         (6) kernel size in seconds
  --ip IP               (0.0.0.0) ip address for OSC client
  --port PORT           (5005) port for OSC client
(.myenv) ➜  applications git:(develop)
```

The help function of `beatcli.py` (Figure 5.2) offers detailed explanations of the input arguments. The real-time audio streaming (Subsection 3.3.2) relies on the Python module `sounddevice`,[3] which receives the first four arguments of the terminal application to specify the desired hardware. The next three arguments control the settings of the beat tracker, including the `tempo` range in BPM to set the pulse level, the `lookahead` (Subsection 3.4.3) in `FRAMES` to compensate for latency, and the `kernel SIZE` in seconds to determine the beat context (Subsection 3.4.2). Last, two arguments configure the network

[3] https://pypi.org/project/sounddevice

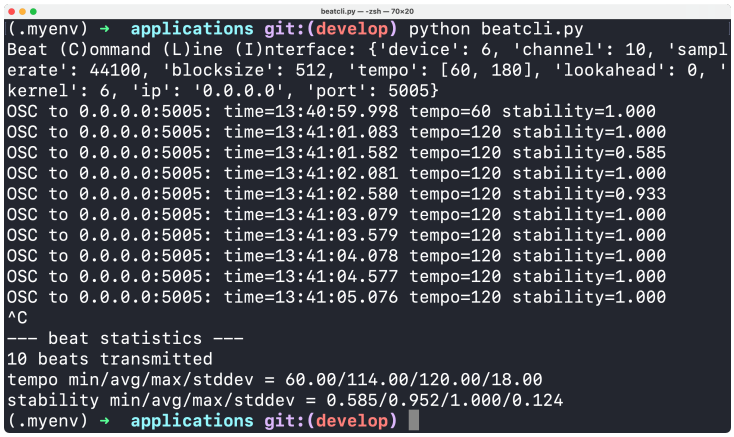


Figure 5.3: The terminal output of the beatcli.py application showing the system in action.

output. `beatcli.py` serves as an open sound control (OSC) [135] server, sending pulse information over the network to a client with a specified `IP` address and `PORT` number.

With Figure 5.3, we illustrate the terminal output of `beatcli.py`, detailing the transmitted pulse information. Upon application launch, an overview of default settings is provided. Below this follows a table of detected beats, each row containing columns with local pulse information. Initially, the `IP` address and `PORT` of the OSC message being sent out are displayed. Subsequently, a `time` value provides a timestamp of when the beat detection (Subsection 3.4.1) occurred. Following this, a `tempo` value indicates the local tempo computed from the inter beat interval (Subsection 3.4.5). Finally, a value for beat stability (Subsection 3.4.4) is displayed. A value of 1.0 represents a beat with a steady tempo and maximum stability, while values close to zero signify a relatively unstable tempo and beat structure.

5.1.2 Real-Time Accompaniment System

In this subsection, we demonstrate a real-world use case of our real-time beat tracking application with audio input from a bass player, as shown in Figure 5.4 and based on the real-time procedure described in Section 3.3.

As the musician performs, the `beatcli.py` terminal application (Subsection 5.1.1) detects beats in real time and sends OSC messages containing local pulse information. These messages are received by a DAW, which uses them to trigger MIDI clips and dynamically adjust the global tempo in the software. This enables synchronized playback, with drums and other sound elements following the bassist's tempo. MIDI clips are configured with probabilistic triggering to introduce variation and produce a more natural, human-like feel.

The stability parameter in `beatcli.py` (Subsection 3.4.4) ensures that samples are only triggered when the beat is stable, effectively controlling volume of the samples and avoiding random playback. The global tempo of the DAW is updated with each beat, maintaining consistent synchronization as the tempo changes to adjust the playback speed of the MIDI clips. Tempo range settings constrain beat detection

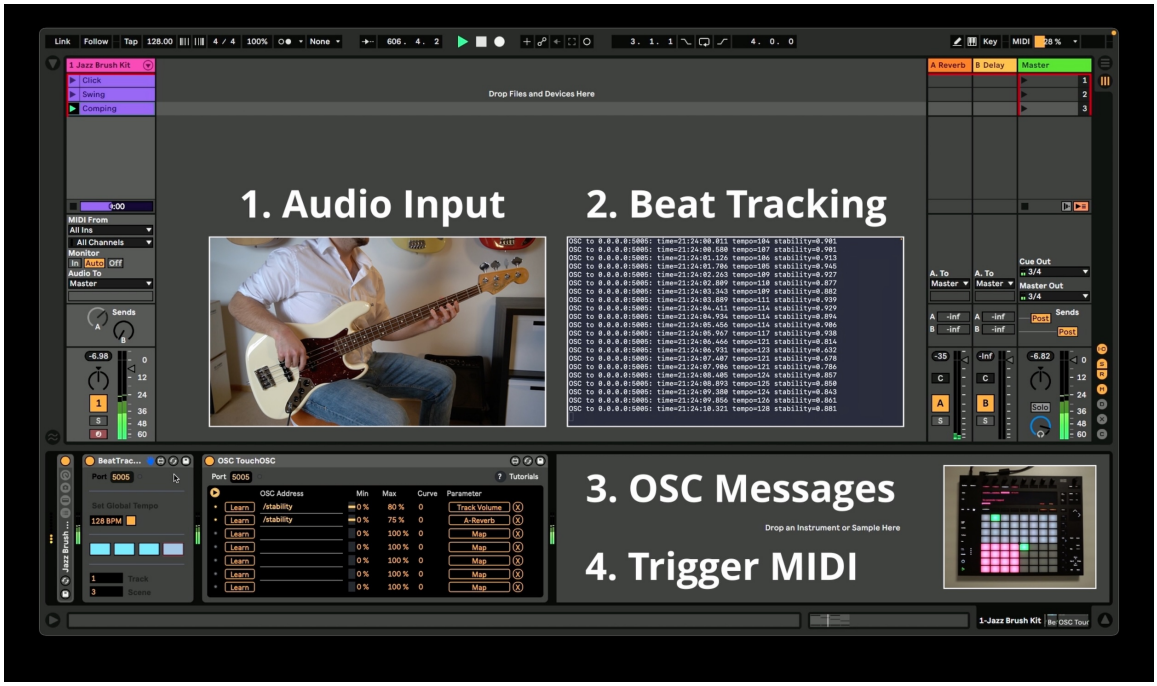


Figure 5.4: A screenshot of the real-time accompaniment system. **(1)** Audio input from a bass player is processed by the `beatcli.py` application, **(2)** which performs real-time beat tracking. **(3)** Detected beats are sent as OSC messages to the DAW, **(4)** where they trigger MIDI clips and dynamically control the global tempo. This setup enables synchronized playback, allowing drums and other sound elements to follow the live performance in real time.

to specific tempo octaves (Subsection 3.4.5), such as normal or double time, accommodating common performance variations. The kernel size parameter (Subsection 3.4.2) adjusts the beat context window, allowing users to trade off between responsiveness and stability of the beat tracking. Finally, the lookahead setting (Subsection 3.4.3) compensates for audio processing and network latency, ensuring that triggered samples align naturally with the live performance. This setup allows for a seamless and expressive musical interaction where the drums dynamically follow the bass player in real time.

5.2 Educational Music Gaming

This section is based on [85]. The first author, Peter Meier, is the main contributor to this article. In collaboration with Simon Schwär, Sebastian Rosenzweig, and his supervisor Meinard Müller, he developed the ideas, designed the game application, and wrote the article. In addition, Peter Meier implemented all the approaches and created the music game prototype.

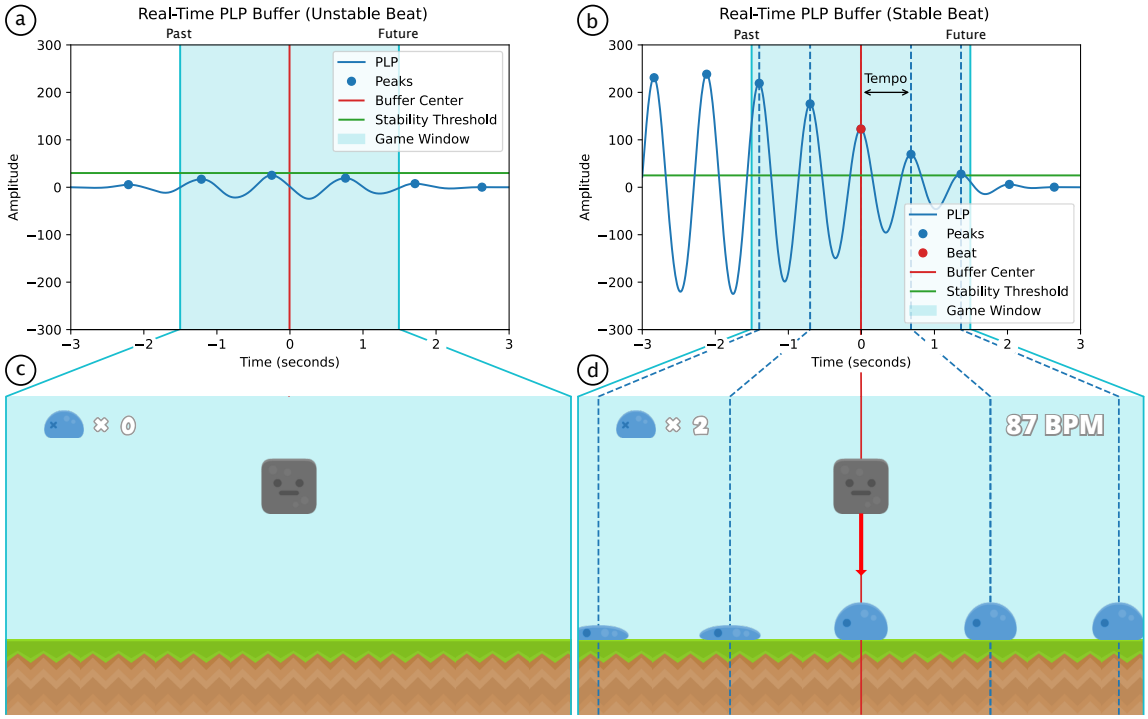


Figure 5.5: Real-Time PLP buffer and world generation of the educational music game "Rock Your Beats." **(a)** Real-Time PLP buffer with an unstable beat. **(b)** Real-Time PLP buffer with a stable beat. **(c)** Empty game world with a rock (black square element). **(d)** Game world with a rock (black square element) hitting moving beat creatures (blue rounded elements).

Based on the real-time PLP procedure detailed in Chapter 3, in this section we present a prototype jump-and-run game named "Rock Your Beats," which is illustrated in Figure 5.5. The player's goal is to tap along in sync with the beat of the music, by pressing a button on a touch screen, keyboard, or gaming controller. In doing so, the player must aim to hit moving "beat creatures" with a dropping rock positioned at the center of the game world. Each beat creature represents one beat, and one point is awarded for each hit. The following subsections provide further details on the real-time generation of the game world (Subsection 5.2.1) and discuss the educational and motivational aspects of the approach (Subsection 5.2.2).

5.2.1 Music-Reactive Game World Generation

The game world is generated in real time from music in the player's environment by using the streamed audio (Subsection 3.3.2) of a microphone input signal. The center of the game world is represented by a rock (see black squared element in Figure 5.5c), which indicates the current point in time (see red line in Figure 5.5a) and thus corresponds to the center of the PLP buffer, as discussed in Subsection 3.3.3. The beat creatures (see the blue rounded elements in Figure 5.5d) are placed at the peak positions of the PLP curve (see the blue dotted peaks in Figure 5.5b), using the beat context, as discussed in Subsection 3.4.2. Therefore, the beat creatures follow the same movement from the right (future) to the left (past) positions

as the peaks of the PLP curve and cross the central player position in sync with the beat of the music, as discussed in Subsection 3.4.1.

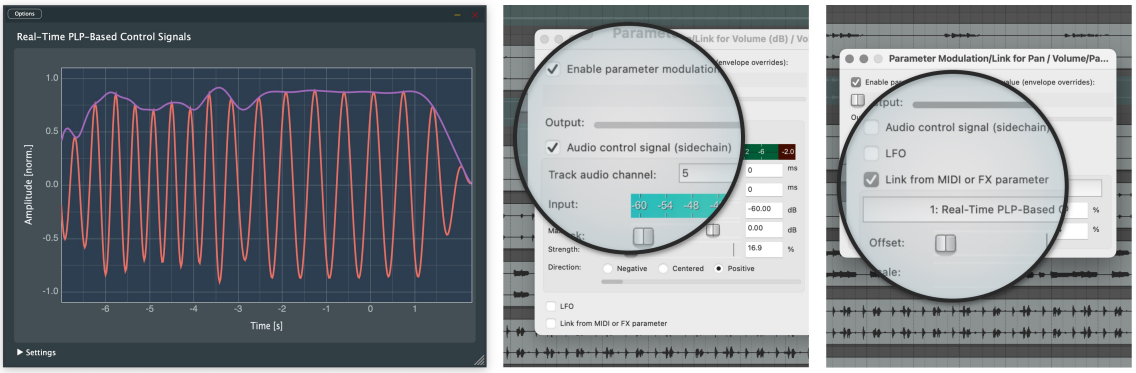
If no stable beat structure is detected in the input signal, the peaks of the PLP curve have a rather low amplitude, as explained in Subsection 3.4.4. If this amplitude is below a certain stability threshold (see green line in Figure 5.5a), no beat creatures are created in the game world (see Figure 5.5c), to avoid unpredictable and noisy beat illustrations. As a consequence, beat creatures only appear if music with a stable beat is played.

The time range that is visually represented in the game, the game window, is closely related to the beat context, as detailed in Subsection 3.4.2. For our example in Figure 5.5, the game window is set to 6 seconds, which means that the game world visualizes the last 3 seconds of past audio beat analysis to predict the next 3 seconds of beat positions. In the same way, the game window can also be chosen smaller to avoid the visualization of inaccurate beat predictions in the distant future and to determine how much visual support the player gets. The inter beat interval (Subsection 3.4.5) can be used to display the current local tempo in the game. Finally, with the beat lookahead (Subsection 3.4.3), the game can be adjusted to compensate for latency that might occur with the game visualization and controller input delays.

5.2.2 Educational and Motivational Aspects

Real-time game world generation with MIR algorithms offers several educational and motivational benefits. First, it encourages players to actively engage with the music in their surroundings, counteracting the common cliché of computer games disconnecting people from the real world. If there is no music playing, no points can be awarded, making the game inherently dependent on the player's environment and musical context. Second, the game provides visual guidance for future beat positions, allowing players not only to feel the beat but also to see it. This visual support can be especially valuable in educational settings, where children and beginners can playfully learn to interact with musical beats and train their rhythmic timing. The size of the game window can be adapted for smaller devices, such as smartphones, by reducing its length, which also determines the amount of visual support provided to the player.

Looking ahead, the prototype can be expanded by incorporating additional MIR algorithms, such as real-time pitch estimation, which will be discussed in Part II (Real-Time Pitch Estimation) of this thesis. These extensions would enable more advanced game mechanics, for example, requiring the player to sing a specific pitch from the current musical scale to achieve a goal. Such developments would further encourage creative engagement with music, allow adaptation to different skill levels, and account for various musical dimensions, making the game a versatile tool for music education and creative exploration.



(a) (b) (c)

Figure 5.6: The prototype implementation and integration in REAPER: **(a)** The audio plugin with visualization of the real-time PLP buffer. **(b)** A sidechain audio signal for effect parameter modulation in REAPER, accessible via the menu `Param > FX parameter list > Param modulation/MIDI link`. **(c)** Utilizing a time-varying parameter value for effect parameter modulation.

5.3 Creative Music Production

This section is based on Section 5 ("Applications") of the paper [89]. The first author, Peter Meier, is the main contributor to this article. Together with Simon Schwär and his supervisor Meinard Müller, he developed the ideas, formalized the mathematical description of the real-time pulse parameters, and wrote the paper. Furthermore, Peter Meier implemented the real-time approach and created a demo visualization. Simon Schwär developed an audio plugin and created mixing examples for the case studies.

In this section, we utilize the real-time PLP procedure introduced in Chapter 3 and especially the control signals derived from it, as described in Chapter 4, to demonstrate creative music production applications. The control signals α-LFO, γ-LFO, β-confidence, and γ-confidence represent musical properties of the analyzed signal which can be utilized in different ways for creative music mixing. To demonstrate some use cases in a practical setting, we developed an audio plugin prototype with JUCE [78] (see Figure 5.6a) that can generate the control signals in real time from any single-channel audio input directly in a DAW. Since many DAWs offer the possibility to use sidechain audio signals to control the modulation of effect parameters (see Figure 5.6b for an example using REAPER [65]), the control signals are provided as separate audio output channels of the plugin. Note that this method requires upsampling of the control signal from the frame rate (i.e., the sampling rate of Δ) to the audio sampling rate. Alternatively, the control signals can be made available as time-varying parameter values of the plugin itself, which can be updated at frame rate and may also be used as a source signal for parameter modulation in REAPER (see Figure 5.6c).

In addition, our implementation includes a real-time visualization of the current α-PLP buffer and its envelope β_{n_0}, which gives insight into the current tempo structure of the analyzed music signal. This provides both a visual indicator for current stability and tracked tempo octave, as well as educational value for exploring PLP settings. For instance, using a microphone input allows for a direct exploration of how parameters such as kernel size or tempo range affect PLP behavior in an intuitive way.

In the following, we will discuss three case studies for the control signals, highlighting musical implications, strengths and limitations of the presented approach for mixing. The examples are further complemented with audio excerpts on our supplemental website.[18]

5.3.1 Case Study 1: Volume Control with Beta-Confidence

Musically, the β-confidence control signal is high when the beat is stable in a musical part and low where the beat is unstable. The α-confidence behaves similarly, but additionally factors in a reduced overall confidence when the buffer read position uses lookahead and is moved into the future (since predicting future beat positions is inherently less reliable). In this way, both confidence signals are suitable for modulating effect parameters that should vary based on the presence of a stable beat. As an example, we may want to change the volume of a pad or drone sound during a bridge part which in popular music often coincides with a change in rhythmic patterns and a reduction of tempo stability. Here, we use the β-confidence to directly control the volume parameter of the channel.

We observed that a low tempo stability in the surrounding musical part often correlates with only short decreases in the confidence control signal. To mitigate this issue, we apply minimum filtering with a variable window size up to multiple seconds, so that a single PLP peak with low amplitude reduces the confidence value for a larger time span. However, since the minimum filter window can only be applied to past confidence values in a real-time setting, this may unintentionally delay the increase of confidence at the beginning of stable sections, so that the window size should be balanced accordingly (see Subsection 5.3.2). After applying such modifications to the confidence, we can also generate a new α-LFO signal by multiplying γ-LFO with the filtered confidence. In addition, while DAWs typically offer various ways to scale the control signal for the desired use case (for example, a confidence of 0 could be mapped to a fader value of -15 dB and a confidence of 1 to 0 dB), a non-linear transformation of the confidence (like exponentiation with an exponent > 0) may further help to achieve the desired effect.

5.3.2 Case Study 2: Noise Gate with Alpha-LFO

The α-LFO signal allows for rhythmically modulating an effect parameter based on the current beat estimates in sections where the beat is stable, while no modulation is done in unstable parts. Musically, this control signal is particularly suitable for effects where the beat-synchronicity is perceptually prominent,

[18] https://www.audiolabs-erlangen.de/resources/MIR/2024-DAFx-RealTimePLP

since in such cases it may be preferable to apply no modulation instead of a modulation that is out of sync with the music. As an example, we can trigger a noise gate using α-LFO, generating a shaker-like effect that is only present in stable beat parts. The lookahead in the real-time PLP buffer additionally allows to adjust the phase of the α-LFO, so that possible latencies of the entire signal processing chain can be compensated for. Furthermore, larger lookaheads can be used to trigger an effect before the next estimated beat, e.g. to generate an off-beat or "reverse snare reverb" effect.

In this context, three limitations of the α-LFO signal may become relevant. First, since a stable beat can only be established after a few pulses have aligned, the α-LFO exhibits a "build-up" at the beginning of sections with a new rhythmic pattern, which is further intensified by minimum filtering the confidence as described in Subsection 5.3.1. Second, it may be desirable to freely choose the tempo octave of the oscillator to control effect parameters on the intended beat division level. This can be influenced to some degree by the tempo range of the local tempo estimation, but a narrow range may lead to unwanted instability when the desired tempo octave is not prominent in the analyzed input signal. Third, many parameter modulations benefit from non-sinusoidal waveforms (e.g., square or sawtooth) to create the desired effect. This may be achieved in future work through a non-linear transformation of the oscillator signal.

5.3.3 Case Study 3: Rhythmic Panning with Gamma-LFO

The γ-LFO is not scaled proportional to beat stability, so that this control signal always utilizes the full modulation range, but may have large and sudden frequency fluctuations in musical parts with low tempo stability. On the other hand, this LFO signal is not affected by the confidence build-up described in Subsection 5.3.2 and can thus be applied for effects where using the full modulation range is always desirable. As an example, we use the γ-LFO to modulate the pan parameter for a channel where the recorded instrument plays in a steady rhythm, causing each played note to alternate between the left and right channels. Since the panning effect is not as perceptually critical w.r.t. timing as for example the shaker in Subsection 5.3.2, some smaller inaccuracies of the γ-LFO synchronization do not negatively affect the outcome, but can rather lead to interesting variations in the mix.

Part II

Real-Time Pitch Estimation

6 A Preliminary Study on Real-Time Pitch Estimation Algorithms

This chapter is based on [86]. The first author, Peter Meier, is the main contributor to this article. In collaboration with Simon Schwär and his supervisor Meinard Müller, he developed the ideas, designed the experiments, and wrote the paper. Furthermore, Peter Meier implemented all approaches and conducted the experiments. Simon Schwär contributed to the preparation of the dataset and developed a dataloader. Gerhard Krump was the supervisor of the main author at the Technische Hochschule Deggendorf.

Real-time pitch estimation is crucial for interactive music applications, including live performance systems, educational tools, and creative interfaces. Accurate and low-latency pitch estimation enhances user interaction in scenarios such as real-time intonation monitoring (see Section 9.1 for an example application) or music-based games (see Section 9.2 for another example). This chapter presents a preliminary study on the suitability of various real-time pitch estimation algorithms focusing on real-time operation, pitch accuracy, and robust voicing detection. We evaluate two well-known and lightweight algorithms, `YIN` [35] and `SWIPE` [21], using common MIR evaluation metrics on a publicly available dataset. These algorithms must provide reliable frequency estimates and distinguish between tonal and non-tonal segments to ensure a smooth and responsive user experience in interactive systems.

In the following sections, we describe the selected real-time pitch estimation algorithms and evaluate their performance. Section 6.1 introduces the algorithms and discusses their properties relevant to real-time applications. Section 6.2 outlines the experimental setup, including dataset, evaluation metrics, and parameter settings. The results are presented and discussed in terms of pitch estimation accuracy, voicing detection, and the effects of different algorithmic variants. Section 6.3 summarizes the main findings and briefly discusses possible directions for future work.

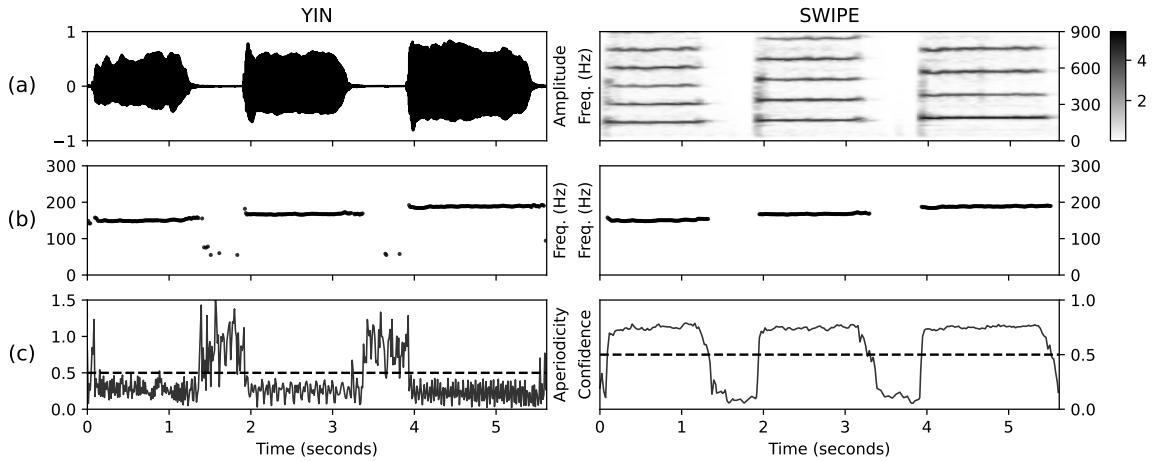


Figure 6.1: An overview of two pitch estimation algorithms evaluated in this study. `YIN`: **(a)** Waveform input. **(b)** Pitch estimation trajectory. **(c)** Aperiodicity for voicing detection. `SWIPE`: **(a)** Spectrogram input. **(b)** Pitch estimation trajectory. **(c)** Confidence for voicing detection.

6.1 Real-Time Pitch Estimation Algorithms

In addition to the general challenges of real-time analysis (see Section 2.2), we now want to take a closer look at the specific challenges of real-time analysis with respect to different pitch estimation algorithms. When we talk about pitch estimation in this chapter, we actually mean fundamental frequency estimation (F_0 estimation), which is often used synonymously with pitch estimation. The basic idea of F_0 estimation is to find the signal's fundamental frequency that correlates with the perceived pitch for each time position [21, 35, 112]. The input to the algorithm is the waveform, spectrogram, or other representation of the signal. The output of the algorithm is the frequency estimate of the input signal and an indication of how confident the algorithm is that the signal is tonal and periodic, called the "confidence value." This confidence value is often used in combination with a threshold to provide a form of voicing detection, for example, to distinguish singing from non-singing signals. Two methods that compute the values on a frame-by-frame basis, and are therefore in principle suitable for real-time applications, are shown in Figure 6.1 and will be discussed in the following section.

`YIN` One of the most well-known F_0 estimation approaches is called `YIN` [35]. The `YIN` algorithm is based on an autocorrelation method and is calculated entirely in the time domain, using the signal's waveform as input, see also Figure 6.1. The algorithm provides accurate pitch estimates for tonal signals, but also incorrectly detects pitches for non-tonal parts of the signal. With respect to real-time applications, `YIN` is relatively simple and can be implemented in an efficient way with low latency. In fact, the offline algorithm can be directly transferred into an online variant, since only one input frame of audio data is required for each output frame of pitch estimation. In addition, `YIN` can be extended with post-processing steps [82], which we would like to investigate more closely for real-time applications in the future. For voicing detection, `YIN` calculates the so-called "aperiodicity" of the input signal. The lower the aperiodicity, the

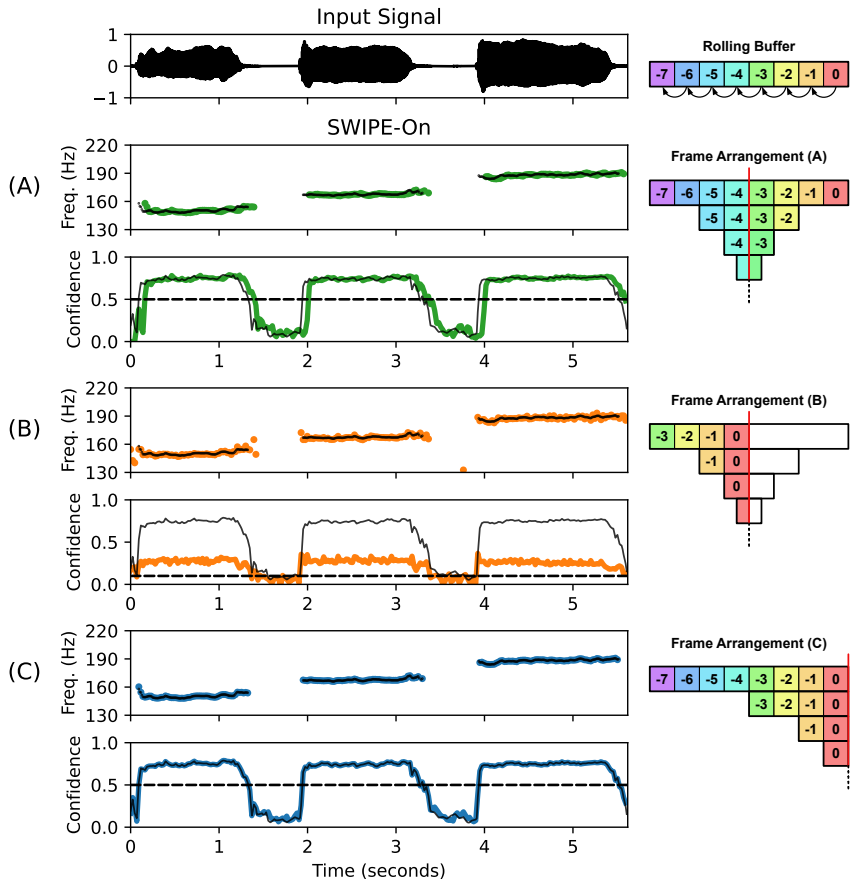


Figure 6.2: Left: Frequency estimates and confidence values of the `SWIPE-On` variants (denoted by **A**, **B**, **C**, and colored in green, orange, blue). Values for `SWIPE-Off` are displayed in black. Confidence thresholds for each variant are indicated by black dashed lines. **Right**: The rolling buffer holds several input frames, symbolized by squares. The numbers indicate the frame index, where 0 is the current frame, -1 is the previous frame, and so on. The frame arrangements for each variant visualize the used signal segment for different window sizes.

higher the probability that the signal is periodic and tonal. This is opposite to the confidence, where lower values indicate a lower probability that the signal is periodic and tonal. To allow a more direct comparison with `SWIPE`, in this chapter we define a confidence = 1 − aperiodicity for `YIN`, which will be used in the following.

`SWIPE` Unlike the `YIN` time-domain algorithm, `SWIPE` is based on spectral correlations in the frequency domain [21]. To this end, `SWIPE` uses multiple pitch candidates with a unique fundamental frequency, each represented by a single pre-computed kernel inspired by a sawtooth waveform. The original offline version of the algorithm (`SWIPE-Off` in the following) transforms the input signal into a time–frequency representation using STFTs. Each pitch candidate requires a different optimal window size for the spectral input, which results in more accurate pitch estimates, but also in a higher computational complexity. Pitch estimates are calculated by correlating the spectrum of each input frame with the set of pitch kernels, finding the kernel with the highest correlation. The amount of correlation is also used as a confidence value

for voicing detection. In the online version of SWIPE, we use a rolling buffer that stores the previous input frames needed to compute all the different window sizes for each frame. This concept allows for three different online variants of SWIPE (SWIPE-On-A/B/C), which are illustrated in Figure 6.2 and described below:

SWIPE-On-A The most obvious arrangement of windows follows the offline algorithm by keeping the spectra perfectly centered and aligned for each point in time. For a causal real-time system, this results in a delayed output signal because the rolling buffer must be filled completely before computation, see (A) of Figure 6.2.

SWIPE-On-B To compensate for this delay, the windows can be filled in such a way that the most recent frames are always in the center of each window, while keeping the alignment between the different window sizes as in SWIPE-On-A. Any samples in the future are set to zero. As illustrated in (B) of Figure 6.2, this causes a reduction in the confidence values. This is because only half-filled windows are available for the spectral correlation with the frequency kernels, lowering the score.

SWIPE-On-C In the third and final online variant, all the different windows are filled with the most recent audio data. This discards a perfectly centered time alignment of frames between different window sizes and thus between F_0 estimates at different frequencies. However, it avoids the problems of (SWIPE-On-A) delayed output and (SWIPE-On-B) reduced confidence, see (C) of Figure 6.2.

6.2 Experiments

For an objective evaluation of the different algorithms, we use the MDB-melody-synth dataset [114]. It consists of 65 different tracks with mainly singing voice melodies derived from the MedleyDB dataset [5]. The MDB tracks were re-synthesized by the method described in [113] to provide perfect ground truth annotations. Thus, MDB-melody-synth is well suited for evaluating real-time algorithms, as the annotations can be easily converted to a frame size of 512 samples and a sampling rate of 44100 Hz, which are also typical real-time audio settings for interactive applications.

In our experiments, we compare a total of six different pitch estimation algorithms: YIN-Off/On, SWIPE-Off, and SWIPE-On-A/B/C. To find a setting that is compatible with all the pitch estimation algorithms as well as the dataset, we run each algorithm with a sampling rate of 44100 Hz, a frame size of 512 samples, a minimum F_0 estimate of 45 Hz, and a maximum F_0 estimate of 1760 Hz. For the evaluation, we measure the quality of the pitch estimation in RPA and RCA, both with 50 cents pitch tolerance. Furthermore, we measure the quality of the voicing detection in VR and VFA, as described in [112]. For all evaluation metrics, we use the reference implementation of mir_eval [107]. In addition,

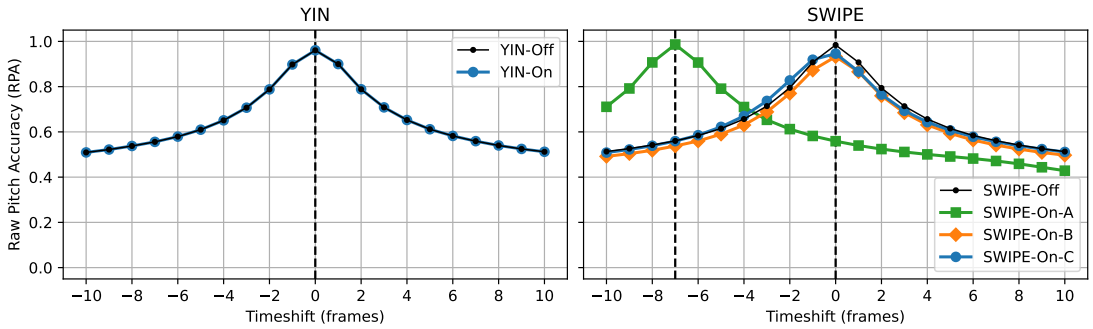


Figure 6.3: Average RPA versus the timeshift (frames) of the estimated frequency values, tested with a pitch tolerance of 50 cents.

Metric	RPA		RCA	
Timeshift	-7	0	-7	0
`YIN-Off`	0.535±0.172	0.926±0.093	0.556±0.166	0.961±0.056
`YIN-On`	0.535±0.172	0.926±0.093	0.556±0.166	0.961±0.056
`SWIPE-Off`	0.554±0.171	0.976±0.092	0.560±0.166	0.984±0.070
`SWIPE-On-A`	0.978±0.091	0.553±0.171	0.986±0.069	0.559±0.166
`SWIPE-On-B`	0.521±0.168	0.909±0.116	0.538±0.161	0.933±0.089
`SWIPE-On-C`	0.556±0.170	0.938±0.101	0.559±0.169	0.946±0.087

Table 6.1: Average raw pitch/chroma accuracies and their standard deviations between different tracks of the dataset for different timeshifts (frames) of the estimated frequency values, tested with a pitch tolerance of 50 cents.

we run all evaluations for different timeshifts (from -10 to 10 frames) and confidence thresholds (from 0.1 to 0.9).

In Figure 6.3 we show the RPA for estimated frequency values shifted in time between -10 and 10 frames. With Table 6.1 we report both RPA and RCA for timeshifts of both -7 and 0 frames. We see that `YIN-Off` and `YIN-On` produce exactly the same accuracy values. `SWIPE-Off` outperforms `YIN-Off` by five percentage points with regard to RPA. When shifted -7 frames, `SWIPE-On-A` achieves similar accuracy as `SWIPE-Off`. For a zero frames timeshift, `SWIPE-On-C` comes closest to `SWIPE-Off` with a drop of 0.038, followed by `SWIPE-On-B` with a drop of 0.067.

In Figure 6.4 we show the VR for different confidence thresholds, and with Table 6.2 we report the values for VR and VFA. `YIN-Off` and `YIN-On` provide exactly the same VR. Furthermore, `SWIPE-Off` has a perfect VR of 1.0 compared to `YIN-Off` with a drop of 0.021, both with similar VFA rates close to zero. `SWIPE-On-B` works only with lower confidence thresholds, where it achieves a similar VR as the other algorithms. The best overall online variant is `SWIPE-On-C` with a VR similar to `SWIPE-Off` and a VFA rate close to zero.

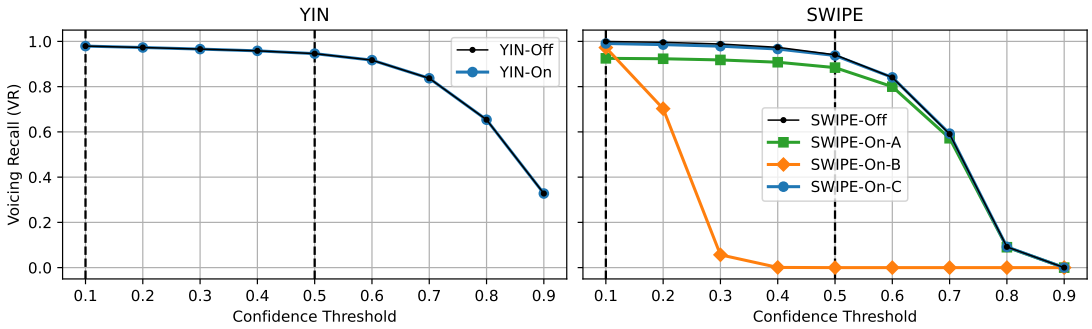


Figure 6.4: Average voicing recall rates versus the confidence threshold.

Metric	VR		VFA	
Confidence	0.1	0.5	0.1	0.5
`YIN-Off`	0.979±0.020	0.946±0.070	0.017±0.021	0.013±0.018
`YIN-On`	0.979±0.020	0.946±0.070	0.017±0.021	0.013±0.018
`SWIPE-Off`	1.000±0.003	0.941±0.088	0.012±0.010	0.000±0.000
`SWIPE-On-A`	0.925±0.054	0.884±0.096	0.131±0.106	0.105±0.089
`SWIPE-On-B`	0.973±0.075	0.000±0.000	0.009±0.010	0.000±0.000
`SWIPE-On-C`	0.991±0.008	0.938±0.087	0.009±0.010	0.003±0.007

Table 6.2: Average voicing recall/false alarm rates and their standard deviations between different tracks of the dataset for different confidence thresholds.

6.3 Conclusions

In this chapter, we compared and evaluated several variants of real-time pitch estimation algorithms for use in interactive music applications, where both the accuracy of pitch estimation and voicing detection are equally important. We have shown that `YIN` transfers well from offline to online and runs very efficiently on the waveform directly, but does not reach the pitch estimation accuracy of `SWIPE`. Compared to `YIN`, `SWIPE` is computationally more complex and requires the additional step of performing multiple STFT operations per frame. Nevertheless, these spectrogram computations are efficient enough to not add noticeable latency to our Python-based test implementation. We presented three different variants for `SWIPE-On`, where (`SWIPE-On-A`) problems with delayed outputs can be solved with timeshifts, and (`SWIPE-On-B`) problems with reduced confidence values can be handled with lower thresholds. We also showed that `SWIPE-On-C` produces the highest overall accuracy for both pitch estimation quality and voicing detection, and is therefore the best option for real-time interactive scenarios.

Based on the insights of this preliminary study, in Chapter 7, we focus on the `SWIPE-On-C` variant and provide a more detailed mathematical description, additional analysis, experiments, and a real-time implementation. Furthermore, in Chapter 9, we discuss practical applications of real-time pitch estimation in Section 9.1 (intonation monitoring) and Section 9.2 (music-based gaming).

7 Real-Time Pitch Estimation Using SWIPE

This chapter is based on [92]. The first author, Peter Meier, is the main contributor to this article. In collaboration with Sebastian Strahl, Simon Schwär, Stefan Balke, and his supervisor Meinard Müller, he developed the ideas, designed the experiments, and wrote the paper. Peter Meier also developed and implemented all approaches and conducted the experiments. Sebastian Strahl contributed to the development and testing of the real-time implementation.

Pitch estimation in real time is essential for a wide range of MIR applications, including intonation monitoring, music education, and interactive systems. Many of these use cases, such as ensemble rehearsals, require low-latency, multi-channel audio processing on resource-constrained devices. While recent approaches with neural networks offer high accuracy, they often fall short in real-time performance due to computational demands. In this chapter, we revisit the well-established SWIPE algorithm and introduce RT-SWIPE, a real-time variant enabled by using causal windowing. We further propose a delay-tolerant evaluation metric that extends RPA to account for algorithmic delays. Experimental results on synthetic signals and multi-track ensemble recordings demonstrate that RT-SWIPE provides a practical balance of latency, accuracy, and efficiency. Although our study focuses on wind orchestra scenarios, the method is broadly applicable to similar real-time settings.

The remainder of the chapter is structured as follows. After introducing the topic and discussing related work in Section 7.1, Section 7.2 introduces the RT-SWIPE algorithm and evaluates its computational efficiency in multi-channel scenarios. Section 7.3 presents systematic experiments on wind ensemble recordings and compares RT-SWIPE to state-of-the-art pitch estimation methods. In addition, we examine the influence of algorithmic delays on the standard RPA metric and propose an extension to address its limitations in real-time settings. Finally, Section 7.4 summarizes our findings and outlines directions for future work.

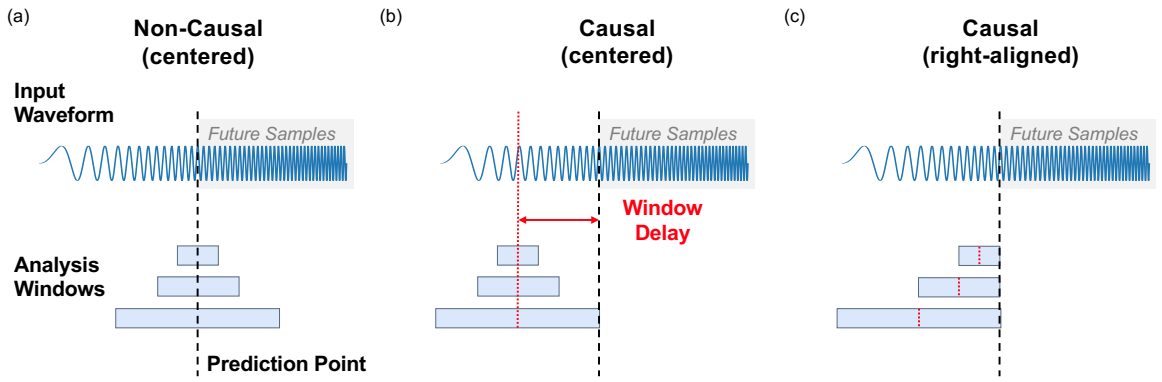


Figure 7.1: Overview of window positioning strategies for real-time audio processing with SWIPE. **(a)** Non-causal, centered windows used in the original (offline) SWIPE algorithm. **(b)** Causal, centered windows with a constant delay for real-time processing. **(c)** Causal, right-aligned windows.

7.1 Introduction

Real-time pitch estimation plays a crucial role in MIR, with applications ranging from intonation monitoring [29] and music education [133] to music scene analysis [49] and interactive gaming [86]. Many of these applications operate on resource-constrained platforms, such as smartphones, and demand low-latency, multi-channel processing [28]. In such contexts, lightweight and real-time capable algorithms such as YIN [35] are often favored over more computationally intensive methods like PYIN [82] and CREPE [66], despite potential trade-offs in estimation accuracy.

The SWIPE algorithm is a well-established method for pitch estimation, recognized for its robustness under real-world conditions [21]. Although not among the most recent developments, SWIPE remains relevant due to its resilience to noise and cross-talk, a critical factor in the context of wind music ensembles [91]. Its use of the FFT ensures computational efficiency, enabling real-time, multi-channel processing even on resource-constrained hardware. While neural network-based approaches may achieve superior pitch accuracy, their computational complexity and hardware demands often limit their applicability in low-latency, real-time scenarios. In addition, the structure of the spectral kernels in SWIPE offers a flexible foundation for further adaptations. Recent developments such as differentiable SWIPE (dSWIPE), which introduces trainable parameters, demonstrate the algorithm's ongoing relevance and adaptability [130]. SWIPE was originally developed for offline pitch estimation. Figure 7.1a illustrates this offline setting, where an input waveform is correlated with a set of analysis windows of varying lengths to estimate the pitch at a given prediction point. This configuration is inherently non-causal, as it requires access to future input samples.

A causal adaptation is shown in Figure 7.1b: By introducing an artificial delay equal to half the length of the longest analysis window, the algorithm can be rendered causal, as it no longer depends on future data. In the illustrated example, the input is a sinusoidal signal with linearly increasing frequency. Due to the uniform shift of all windows, pitch estimates for higher frequencies (which rely on shorter windows)

may suffer from reduced accuracy, as the analysis does not capture the most recent signal content. This limitation can introduce a systematic bias toward lower frequencies, which is particularly problematic in musical contexts involving glissandi or rapid note changes.

As shown in Figure 7.1c, this issue can be mitigated by right-aligning all analysis windows with respect to the prediction point. This strategy ensures that each window captures the most up-to-date input samples. Similar approaches were used before, as demonstrated in [14], which explored frequency dependent latency for the Constant-Q Transform (CQT) [18, 19]. While the overall delay (determined by the longest window) remains unchanged, the responsiveness of shorter windows improves substantially. Since these windows correspond to higher frequencies, which typically exhibit faster temporal variation, the alignment helps preserve estimation accuracy during fast transitions. In contrast, longer windows associated with lower frequencies are less affected, as low-frequency content tends to vary more slowly.

In our research efforts, we aim to apply pitch estimation in ensemble settings, e. g., wind ensembles or orchestras. In such contexts, low latency is essential, as many channels must be processed simultaneously in real time. Furthermore, the presence of variable acoustics and interference from other instruments poses additional challenges, establishing ensemble environments as demanding yet valuable testbeds for real-time pitch estimation. Our previous studies identify SWIPE as the most suitable approach for these scenarios. Compared to CREPE, it is significantly less computationally demanding, and it offers greater robustness to cross-talk than YIN [91].

This chapter continues this line of research by adapting SWIPE to operate in a causal manner, making it suitable for real-time applications. We also conduct systematic experiments on ensemble recordings, closely aligned with our target use case. During our evaluation, we observed that RPA exhibits limitations, particularly when applied to real-time scenarios. To address this, we propose an extension to the RPA metric that accounts for timing tolerances. While our experiments focus specifically on ensemble music, the underlying concepts and findings are applicable to a broader range of real-time pitch estimation tasks.

7.2 Real-Time SWIPE

Our real-time approach builds upon the original SWIPE algorithm [21]. In this section, we outline the key components of SWIPE that are relevant to our real-time modifications. The SWIPE algorithm estimates pitch by examining the harmonic structure of a sound. It utilizes pitch candidate kernels derived from a sawtooth waveform and correlates these with the spectrum of an input signal to identify the best match. The candidate with the highest correlation score is selected as the estimated pitch.

The SWIPE algorithm employs different window sizes to accurately estimate the short-time spectrum of an input audio signal. For a given set of predefined pitch candidates F_i, each represented by a frequency-domain kernel, we determine the optimal window sizes N_i that maximize the overlap between these kernels and the spectral lobes. These window sizes are calculated as follows:

$$N_i = \frac{8 \cdot F_s}{F_i}, \tag{7.1}$$

where F_s stands for the sampling rate of the input signal in Hz. However, instead of computing individual spectrograms for each pitch candidate, SWIPE approximates this process by calculating spectrograms for a limited set of window sizes that are powers of two. This approach requires interpolation between window sizes, but significantly reduces computational costs [21]. To generate the spectral representation, SWIPE utilizes window sizes tailored to the pitch range. Let $F_{\min}$ and $F_{\max}$ denote the minimum and maximum expected frequencies in Hz, respectively. We define:

$$a = \left\lfloor \log_2 \left(\frac{8 \cdot F_s}{F_{\max}} \right) \right\rfloor, \quad b = \left\lceil \log_2 \left(\frac{8 \cdot F_s}{F_{\min}} \right) \right\rceil. \tag{7.2}$$

From these, the smallest and largest window sizes are calculated as:

$$N_{\min} = 2^a, \quad N_{\max} = 2^b. \tag{7.3}$$

Using these boundaries, we define a set K that includes all power-of-two window sizes used for spectrogram computation:

$$K = \{N_{\min}, 2 \cdot N_{\min}, ..., N_{\max}\} = \{2^a, 2^{a+1}, ..., 2^b\}. \tag{7.4}$$

For more details on the original method, we refer to [21].

7.2.1 Real-Time Audio Processing

In real-time audio processing, data is managed block-wise. These blocks, or frames, contain $H \in \mathbb{N}$ samples each and do not overlap. We refer to H as the hop size. When using SWIPE for real-time audio processing, three key considerations arise: (1) Real-time processing follows strict time constraints. The computation time, or **processing latency**, must not exceed a single frame period T_{frame}, given by:

$$T_{\text{frame}} = \frac{H}{F_s}. \tag{7.5}$$

This constraint is particularly relevant for multi-channel scenarios, where the processing latency may scale with the number of channels. (2) A single audio frame is typically smaller than the maximum window size $N_{\max} \in \mathbb{N}$ required for estimation. To address this, we need to implement audio buffering. (3) In real-time processing, only past data is accessible. Since analysis windows are usually centered, as illustrated in

Figure 7.1b, estimates are delayed by half the maximum window size $N_{\max}$, introducing **algorithmic delay**.

To manage larger window sizes, we use a rolling buffer B that efficiently handles continuous audio input by cyclically adding new frames and discarding the oldest ones, defined as follows:

$$B \in \mathbb{R}^{C \times N_{\max}}. \tag{7.6}$$

$C \in \mathbb{N}$ is the number of audio channels, and $N_{\max}$ is the largest window size (see Equation 7.2 and Equation 7.3).

For each new frame of audio input, an estimate is generated through a sequence of computations that must be completed within a single frame period T_{frame}. The following steps are performed:

(a) Update buffer B with the latest frame of audio input.

(b) For each window size $N_i \in K$, extract the corresponding data from the buffer according to the window positioning (see Subsection 7.2.2).

(c) Apply a Hann window to each window.

(d) Compute the DFT, vectorized over all C channels to ensure computational efficiency.

(e) Follow the original method to correlate the resulting spectra with pitch candidate kernels and select the candidate with the highest score as the estimated pitch.

7.2.2 Adjustable Window Positioning

In Figure 7.2a, we illustrate a more detailed view of causal centered windows as used in the RT-SWIPE method. All windows are not centered around the prediction point (compare to the non-causal case in Figure 7.1a); instead, each window extracts data from the center of buffer B, containing only past samples. Consequently, estimates are delayed by half the maximum window size $N_{\max}$, even though more recent samples could be used for smaller windows (higher frequencies).

To address this, we propose a right-aligned window positioning (Figure 7.2b), where each window uses the most recent audio samples available in buffer B. This reduces the delay, especially for small window sizes, thereby affecting the expected algorithmic delay for high pitches. To interpolate between centered and right-aligned window positioning, we introduce a normalized delay factor $\Delta \in [0, 1]$, where $\Delta = 1$ corresponds to centered and $\Delta = 0$ to right-aligned positioning.

For each delay factor Δ, the corresponding delay size D in samples is defined as

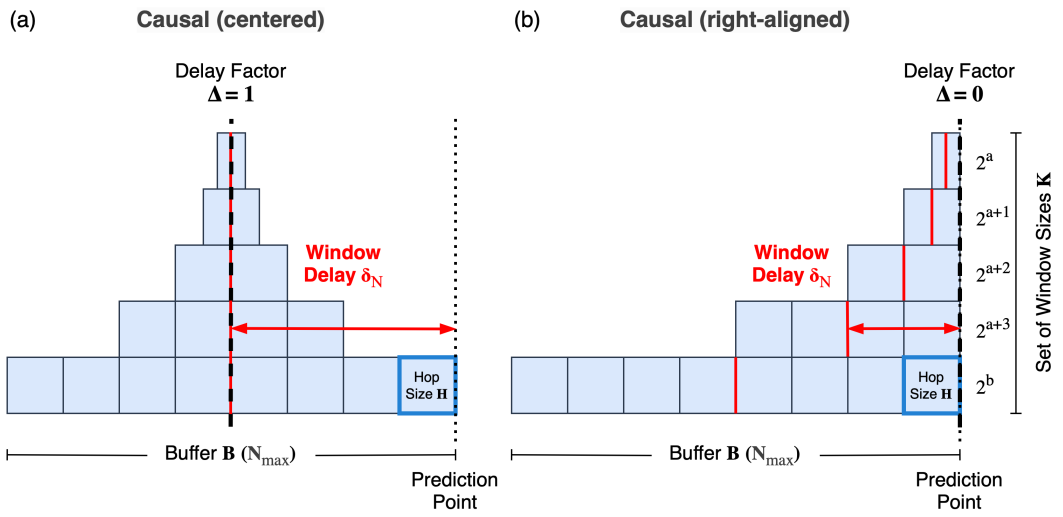


Figure 7.2: Detailed view of adjustable window positioning with delay factor Δ for RT-SWIPE: **(a)** Centered windows share a common maximum delay. **(b)** Right-aligned windows use the latest audio data for estimation, each with individual delays.

$$D(\Delta) = \left\lfloor \Delta \cdot \frac{N_{\max}}{2} \right\rfloor, \tag{7.7}$$

which describes the position at which the windows are intended to be centered, i.e., the position of the black dashed line in Figure 7.2. Depending on the individual window sizes N_i, we further introduce window delays (given in samples)

$$\delta_{N_i} = \max\left(\frac{N_i}{2}, D\right), \tag{7.8}$$

which represent the positions at which the windows are actually centered, considering that only past samples are available. These center positions are illustrated by red lines in Figure 7.2.

For right-aligned window positioning ($\Delta = 0$), the window delays δ_{N_i} vary, introducing a pitch-dependent delay. Low pitches corresponding to large windows have large delays, while high pitches with small windows have small delays. In contrast, centered window positioning ($\Delta = 1$) provides a frequency-independent delay determined by the maximum window size $N_{\max}$.

7.2.3 Multi-Channel Efficiency

For our research and field studies on wind music ensembles, we require not only real-time pitch estimation but also the ability to handle multiple input channels simultaneously. To this end, we investigate the multi-channel efficiency of the RT-SWIPE implementation and test how many input channels can be processed on consumer-level hardware.

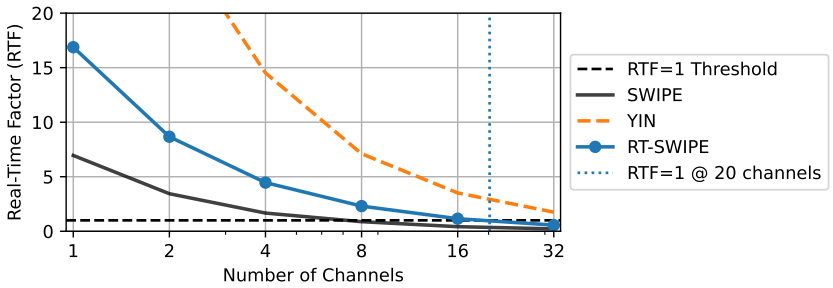


Figure 7.3: Efficiency of `RT-SWIPE` compared to `YIN` and `SWIPE`: Real-Time Factor (RTF) over number of channels.

As illustrated in Figure 7.3, we measure the RTF as the number of input channels increases, identifying the maximum number of channels for which the RTF remains 1. An RTF smaller than 1 indicates computations take longer than a frame period, making it unsuitable for real-time processing. In contrast, an RTF greater than 1 means computations are faster than required, making it suitable for real-time applications. For our efficiency experiment, we conduct tests on a Mac Mini 2023 equipped with an Apple M2 Chip and 16 GB of random access memory (RAM). We compare `RT-SWIPE` with an offline implementation of `SWIPE` [111], which is not optimized for frame-based processing and, therefore, serves as a lower performance bound. For an upper performance benchmark, we use the `YIN` implementation from librosa [83], a frame-based algorithm known for its speed and efficiency due to its autocorrelation method.

While the `YIN` implementation can process up to 38 channels on our test computer in real time, the offline `SWIPE` implementation can handle up to 7. In contrast, `RT-SWIPE` achieves a solid balance by supporting up to 20 channels in parallel. This highlights the trade-off between computational efficiency and robustness in pitch estimation, with `SWIPE`-based methods generally offering greater reliability in noisy, real-world scenarios compared to `YIN` [91].

7.3 Experiments

Dataset Our experiments are performed on the `ChoraleBricks` dataset, a compilation of 193 multi-track recordings featuring wind instruments such as brass and woodwinds [1]. The dataset includes performances of 10 different chorale pieces, with soprano, alto, tenor, and bass parts played by different instruments. Each instrument was recorded separately but synchronized using a conducting video. In addition to the multi-track recordings, `ChoraleBricks` includes F_0 annotations, interactively generated using *Sonic Visualiser* (v5.0.1) [22] and the *pYIN VAMP plugin* (v3) [82]. These annotations were verified through sonification methods [99] and manually corrected as needed. The fundamental frequency (F_0) of reference annotations range from 38.19 Hz (played by a tuba) to 1250.97 Hz (played by a flute).

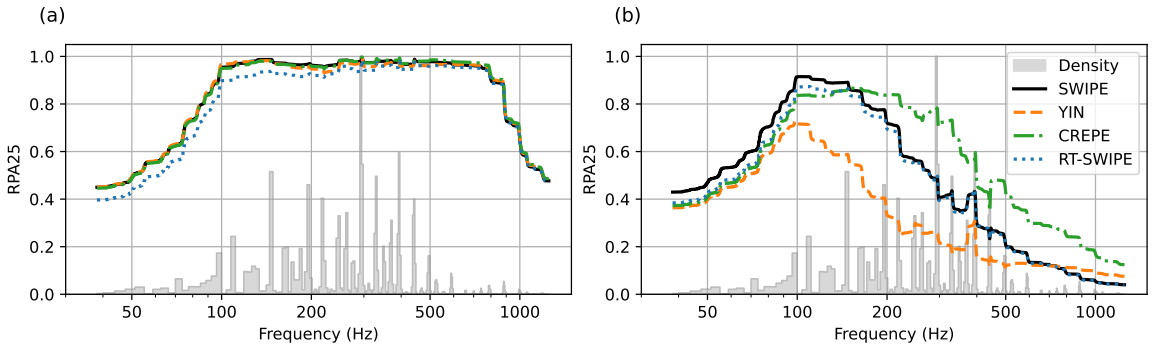


Figure 7.4: RPA with a 25-cent tolerance over frequency on the ChoraleBricks dataset for SWIPE, YIN, CREPE, and RT-SWIPE. **(a)** Instruments without cross-talk. **(b)** Instruments with cross-talk: target instrument signals are mixed with a randomly selected interfering instrument from a different voice, using a signal-to-noise ratio (SNR) of 5 dB. Background bars show the pitch distribution in the ChoraleBricks dataset.

Experimental Setup In our experiment, we utilize RT-SWIPE with right-aligned window positioning ($\Delta = 0$). We compare its performance against the original SWIPE method as implemented by the Python library libf0 [111], and YIN [35], as implemented in librosa.[19] We also compare it with CREPE [66], using the official implementation available on GitHub.[20] The experiments are conducted with a sample rate $F_s = 44.1$ kHz and a hop size $H = 512$ samples. All estimators operate within a frequency range from note C1 (approximately 32.70 Hz) to note A6 (1760 Hz), effectively covering the entire range of the ChoraleBricks dataset. The cent resolution is configured to 10 cents. For the pitch evaluation, we employ RPA with a tolerance of 25 cents.

7.3.1 Results

In Figure 7.4a, we present the frequency-dependent evaluation results of our experiment. For each estimated frequency, we check if it matches the reference within the cent tolerance. We sort this binary data (1 = correct; 0 = not correct) by annotation frequency and apply a Hanning window-based convolution to smooth the results and achieve RPA values over frequency. The RPA values for SWIPE are illustrated by the black solid line, those for RT-SWIPE are shown by the blue dotted line, CREPE is represented by the green dash-dotted line, and YIN is depicted with the orange dashed line. The step function, with a gray shaded area, indicates the density of F_0 reference annotations, helping to understand the frequency distribution in the ChoraleBricks dataset. From this, we identify that most notes are around 300 Hz, whereas notes at 50 Hz or 1 kHz are rarely played.

Across the dataset, SWIPE achieves an overall RPA of 0.960, while YIN records an RPA of 0.952, and CREPE reaches 0.961, indicating very similar performance among these algorithms. These results align with findings from previous studies [91]. However, RT-SWIPE shows a lower RPA of 0.931.

[19] We specifically used commit ebd878f, which includes recent updates and bug fixes for YIN and PYIN.
[20] https://github.com/marl/crepe.

As illustrated in Figure 7.4a, the decrease in RPA for RT-SWIPE is frequency-dependent. All algorithms perform similarly above 500 Hz but diverge at lower frequencies. For instance, at 200 Hz, the RPAs for SWIPE, YIN, and CREPE are 0.963, 0.948, and 0.959, respectively, compared to 0.922 for RT-SWIPE. At 50 Hz, the differences become more pronounced: SWIPE, YIN, and CREPE have RPAs of 0.490, 0.492, and 0.486 respectively, while RT-SWIPE falls to 0.434.

Figure 7.4b shows the results of the various estimation methods in a cross-talk scenario. Here, target instruments are mixed with randomly selected interfering instruments from a different voice (soprano, alto, tenor, bass), using a SNR of 5 dB. As observed in our previous study [91], YIN is most sensitive to interference from cross-talk. CREPE achieves the highest overall accuracy, while SWIPE offers a favorable balance between performance and computational efficiency. Notably, all approaches exhibit increased sensitivity to cross-talk in frequency regions above 200 Hz.

The observed deviations of RT-SWIPE arise from a mismatch between real-time estimation and the RPA metric, which assumes perfect temporal alignment between estimates and reference annotations. In real-time settings, however, causal windowing introduces algorithmic delays, causing accurate but slightly delayed estimates to be penalized as incorrect.

7.3.2 Results with Time Tolerance

In the following, we illustrate the algorithmic delays introduced through the causal windowing with a synthetic test signal. In Figure 7.5a, the reference F_0 trajectory of the test signal is shown as a black line. This trajectory features sinusoidal oscillations around a base frequency F_{base}, resembling a vibrato pattern. The modulation has a frequency of $F_{\text{mod}} = 1$ Hz and an amplitude of $A_{\text{mod}} = 0.2 \cdot F_{\text{base}}$. The test signal includes four segments, each two seconds long, where the base frequency F_{base} doubles sequentially from 50 Hz to 100 Hz, 200 Hz, and finally 400 Hz. We sonify this F_0 trajectory using the `libsoni` [99] toolbox with the function `sonify_f0`, utilizing 20 partials with individual amplitudes of $1/20$, at a sampling rate F_{s} of 44.1 kHz to mimic a harmonic signal.

The test signal is analyzed with two delay factors: $\Delta = 1$ (centered) and $\Delta = 0$ (right-aligned). The parameters are as follows: $F_{\text{min}} = 30$ Hz, $F_{\text{max}} = 700$ Hz, $F_{\text{s}} = 44.1$ kHz, and $H = 512$ samples. The centered window positioning with a delay factor of $\Delta = 1$ is illustrated by the blue dotted line in Figure 7.5a. The estimated trajectory closely resembles the original reference trajectory, although there is a constant algorithmic delay of 186 ms. This delay stems from the window delay associated with the largest window (N_{max}).

In contrast, right-aligned window positioning with $\Delta = 0$ is illustrated by the red dashed line in Figure 7.5a. Like the center-aligned configuration, this trajectory closely follows the reference but exhibits a frequency-dependent delay rather than a constant one. Specifically, the delay decreases with increasing pitch: at

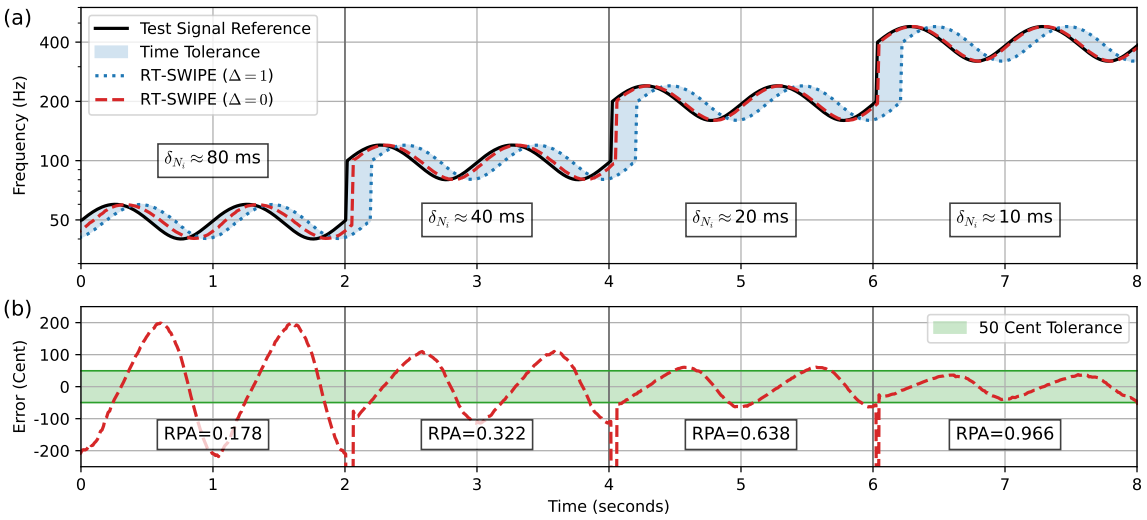


Figure 7.5: (a) Reference annotation of a vibrato test signal (black line) with estimates for different delay factors Δ. **(b)** Error of RT-SWIPE (Δ = 0) estimates compared to the reference in cents, with corresponding RPA values.

50 Hz, the delay is approximately 80 ms; at 100 Hz, it drops to around 40 ms; at 200 Hz, to about 20 ms, and at 400 Hz, to roughly 10 ms.

In Figure 7.5b, the red line represents the estimation error in cents between the reference and RT-SWIPE with right-aligned window positioning (Δ = 0). The green shaded area indicates a 50-cent tolerance used for evaluating the RPA. For the 400 Hz segment (seconds 6 to 8), the error curve remains well within the tolerance band, resulting in a high RPA of 0.966. At 200 Hz (seconds 4 to 6), the RPA drops to 0.638, and further declines to 0.322 for the 100 Hz segment (seconds 2 to 4). The lowest RPA of 0.178 is observed in the 50 Hz segment (seconds 0 to 2), where large portions of the error curve fall outside the tolerance range.

This example indicates that strict frame-based evaluation, like RPA, may not be ideal for evaluating real-time signals with delayed estimates. Similar to how cent tolerance considers pitch estimates within a certain pitch range as correct (see the green area in Figure 7.5b), managing delayed estimates also requires a time tolerance (see the blue area in Figure 7.5a).

To address this limitation, we propose extending the RPA metric by incorporating a time tolerance in addition to the usual cent-based pitch tolerance. For each reference frame, we evaluate not only the pitch estimate at the same time frame but also those in subsequent frames within a specified time window $\tau \in \mathbb{R}$ (given in ms). If any of these estimates fall within the allowed cent tolerance (e. g., ±25 cents), we count the reference frame as correctly estimated. Since we are dealing with real-time systems, the tolerance window includes only future frames, consistent with the causal nature of the application.

Figure 7.6a and Figure 7.6b illustrate the effect of varying time tolerances on our evaluation, shown for scenarios without and with cross-talk, respectively. At $\tau = 0$ ms, the result matches the original frame-based evaluation from Figure 7.4. As τ increases, RPA values improve, especially at lower frequencies where algorithmic delays are more pronounced. With $\tau = 23.2$ ms, the RPA aligns more

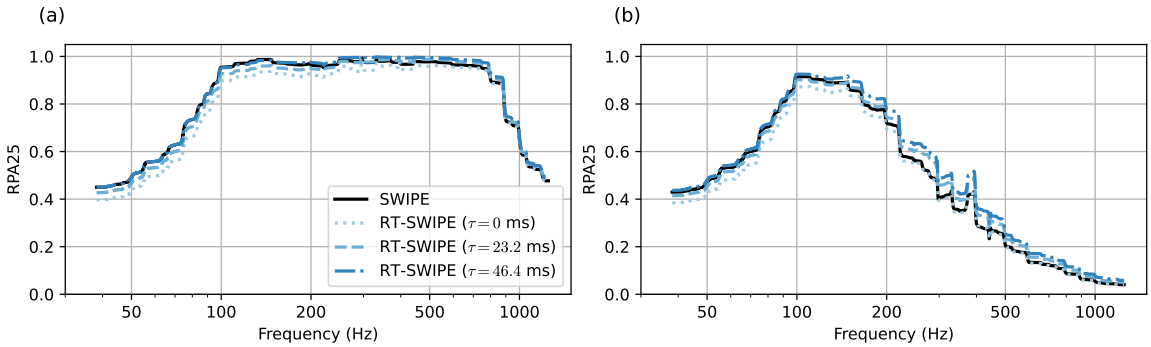


Figure 7.6: RPA with a 25-cent tolerance over frequency on the `ChoraleBricks` dataset for `RT-SWIPE` at varying time tolerances τ, using `SWIPE` as a reference. **(a)** Instruments without cross-talk. **(b)** Instruments with cross-talk: target instrument signals are mixed with a randomly selected interfering instrument from a different voice, using a SNR of 5 dB.

closely with the offline baseline (`SWIPE`). When τ is increased to 46.4 ms, the RPA results are nearly identical to the `SWIPE` baseline at lower frequencies and slightly higher at higher frequencies. This indicates that estimation accuracy degradations are a result of delayed estimates, a problem that is not taken into account by standard RPA measures.

7.4 Conclusions

In this chapter, we presented `RT-SWIPE`, a real-time adaptation of the `SWIPE` algorithm with adjustable window positioning to reduce pitch estimation delays. Our analysis shows that it supports real-time processing of up to 20 channels on consumer hardware. Experiments with synthetic signals highlight that while `RT-SWIPE` reduces algorithmic delay, frame-based metrics like RPA can misrepresent delayed yet accurate estimates. To address this, we proposed a time-tolerant extension to RPA. Tests on real-world data confirm that `RT-SWIPE` preserves the accuracy of the original method while enabling real-time use with controlled delay characteristics.

In future work, we plan to employ `RT-SWIPE` as the algorithmic backbone for applications in ensemble music and interactive singing games. One promising direction is the development of tools that provide musicians with objective intonation feedback to support and enhance auditory skills, as demonstrated in Section 9.1. Another is its integration into interactive music games, enabling real-time evaluation of player performance, as discussed in Section 9.2. Beyond these application-oriented goals, we also aim to further advance the algorithm itself. In particular, `dSWIPE` [130], a differentiable extension of `SWIPE`, presents a compelling research avenue: By enabling the training of spectral templates tailored to individual instruments, it may enhance robustness in acoustically challenging conditions. This is especially relevant in scenarios with significant cross-talk and interference between sources; challenges that will be examined in detail in the following Chapter 8.

8 Analyzing Pitch Estimation Accuracy in Cross-Talk Scenarios

This chapter is based on [91]. The first author, Peter Meier, is the main contributor to this article. In collaboration with Stefan Balke and his supervisor Meinard Müller, he developed the ideas, designed the experiments, and wrote the paper. Furthermore, Peter Meier implemented all approaches and conducted the experiments.

Intonation accuracy is crucial for wind instrument ensembles, where pitch deviations affect harmonic coherence. MIR techniques, particularly pitch estimation, offer potential for real-time intonation monitoring. However, in natural ensemble settings, microphone cross-talk can compromise pitch accuracy. In this chapter, we systematically investigate the impact of cross-talk on pitch estimation for wind instruments using the `ChoraleBricks` dataset, which contains multi-track recordings of isolated choral performances. By simulating cross-talk scenarios with Gaussian noise, single- and multi-instrument interference, we assess the robustness of lightweight, real-time capable estimators like `YIN` and `SWIPE` against more advanced methods like `PYIN` and `CREPE`. Our results show that pitch estimation accuracy declines significantly below an SNR threshold of 15 dB. To address this, we identify instrument-specific challenges and propose frequency filtering to mitigate cross-talk interference. These findings inform the development of robust, real-time intonation monitoring systems for wind ensembles, with applications in music education, performance analysis, and rehearsal optimization.

This chapter is organized as follows: We begin with an introduction to the topic and related work in Section 8.1. Section 8.2 describes our application scenario and dataset. Section 8.3 presents baseline results, while Section 8.4 explores three interference types: Gaussian noise, single-instrument, and multi-instrument cross-talk. Finally, Section 8.5 summarizes our findings and suggests future directions. Audio examples and additional materials are available on a dedicated website.[21]

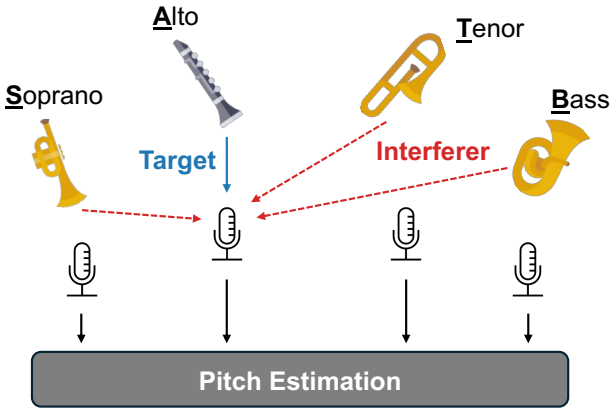


Figure 8.1: Cross-talk in wind instrument quartets: Each microphone records its designated instrument's sound (target) and unintended signals from other instruments (interferer).

8.1 Introduction

Cross-talk is a common challenge in ensemble music recording, occurring when a microphone captures not only its intended instrument but also unintended sounds from other sources, as illustrated in Figure 8.1. This interference introduces artifacts in recorded audio, significantly affecting tasks such as pitch estimation and intonation monitoring. Previous research has explored various strategies to mitigate cross-talk. Prätzlich et al. [105] proposed methods to reduce interference in multi-channel recordings, while Seipel and Lerch [120] simulated cross-talk using anechoic orchestral recordings to address multi-track recording challenges. Maher and Beauchamp [79] examined pitch estimation in noisy environments, focusing on instrument-specific noise such as flute turbulence. Similarly, Singh et al. [122] assessed the noise robustness of pitch estimation, using their *DeepF0* method to evaluate accuracy under varying levels of accompaniment noise.

In this work, we investigate pitch estimation accuracy in cross-talk scenarios, specifically for wind instruments, with applications in intonation monitoring in mind. We evaluate traditional methods such as `YIN` [35] and `SWIPE` [21], along with more advanced techniques like `PYIN` [82] and `CREPE` [66]. Our experiments reveal that `YIN` and `SWIPE` achieve a good balance of robustness, efficiency, and real-time capability for this application.

The main contributions of this chapter are as follows. First, we systematically investigate cross-talk for various pitch estimation approaches, revealing its significant impact on accuracy and establishing a 15 dB SNR threshold for reliable detection. Second, we leverage a novel dataset of 13 wind instruments to enable systematic experiments in cross-talk scenarios—a setting rarely explored in MIR research. Third, we provide insights into instrument-specific challenges and propose practical solutions, such as frequency filtering, to improve algorithmic robustness. Although these topics are not entirely new, this study highlights wind instruments as an important research area and is a step toward creating tools that benefit wind instrument players in real-world applications.

[21] `https://www.audiolabs-erlangen.de/resources/MIR/2025-SMC-PitchCrosstalk`

8.2 Application and Dataset

In this section, we present the application scenario that motivates our experiments (see Subsection 8.2.1). We also explore the dataset used in our study (see Subsection 8.2.2), providing key statistics on the available instruments.

8.2.1 Application

In wind instrument ensembles, real-time pitch and intonation monitoring systems could provide valuable support during rehearsals, recordings, and live concerts. By analyzing multiple microphone channels simultaneously, these systems can offer conductors and musicians not only insight into individual instrument intonation but also into ensemble tuning and any pitch drift during performances. These tools can help quickly resolve intonation issues by identifying the specific instruments that need adjustment. While real-time pitch monitoring tools, such as the Python application Pytch,[22] are already utilized in singing applications, cross-talk is usually mitigated using special larynx microphones. With the microphone setups typically used for wind instruments, cross-talk on individual instrument microphones is a potential issue that we aim to investigate through our experiments.

8.2.2 Dataset

For our experiments, we use the `ChoraleBricks` dataset [1] to simulate the cross-talk scenario. This dataset includes multi-track recordings of ten four-part chorales, with each part, soprano (`S`), alto (`A`), tenor (`T`), and bass (`B`), recorded separately but in sync using a conducting video. Table 8.1 provides an overview of the wind instruments in the dataset. For detailed track distribution across the chorales, refer to [1]. This dataset is ideal for comparing pitch estimation among different instrument tracks and allows for creating mixes with diverse instrument combinations. For instance, one ensemble might consist solely of brass instruments like two trumpets, a baritone, and a tuba, while another mix could combine brass and woodwinds like trumpet, clarinet, tenor saxophone, and tuba.

Besides the multi-track recordings, `ChoraleBricks` includes F_0 annotations, generated interactively using *Sonic Visualiser* (v5.0.1) [22] and the *pYIN VAMP plugin* (v3) [82]. These annotations were verified using sonification methods [99] and manually corrected if necessary. For the tuba (`tba`), a salience-based F_0 estimator was used [1, 94].

Figure 8.2 shows the MIDI pitch distribution for each instrument in the dataset, sorted by their annotated F_0 median frequency. This order will be maintained in later analyses. The instruments roughly fall into two groups based on pitch distribution: Group 1, from oboe (`ob`) to English horn (`eh`), and Group 2, from French horn (`fho`) to trombone (`tb`). Flutes (`fl`) play an octave higher than Group 1, reaching the dataset's

[22] `https://github.com/pytchtracking/pytch`

ID	Instrument	Type	Part	#
fl	Flute	W	S	10
ob	Oboe	W	S	10
as	Alto Saxophone	W	SA	20
tp	Trumpet	Br	SA	20
fh	Flugelhorn	Br	SA	20
cl	Clarinet	W	SA	20
eh	English Horn	W	A	10
fho	French Horn	Br	AT	11
bar	Baritone	Br	SATB	24
bs	Baritone Saxophone	W	TB	18
bcl	Bass Clarinet	W	TB	12
tb	Trombone	Br	TB	8
tba	Tuba	Br	B	10

Table 8.1: IDs, types, parts, and track counts per instrument in the ChoraleBricks dataset. Instrument families: W (woodwind), Br (brass). Part categories: S (soprano), A (alto), T (tenor), B (bass).

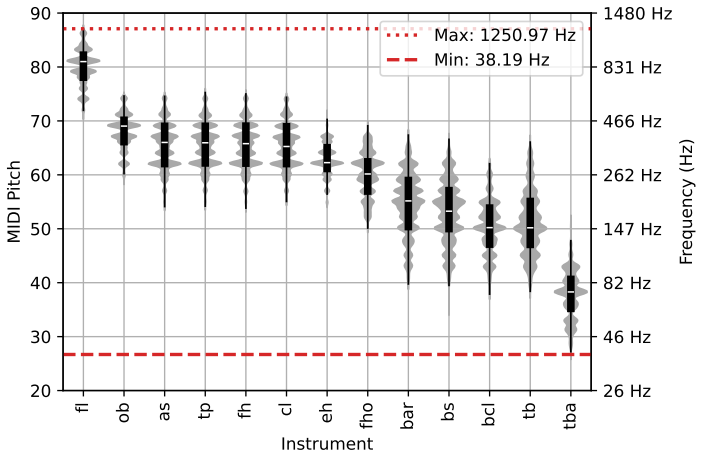


Figure 8.2: MIDI pitch distribution of ChoraleBricks by instrument, based on F_0 reference annotations, sorted by median frequency (white markers). Red dashed lines show the dataset's minimum and maximum frequencies.

maximum frequency of 1250.97 Hz, while tubas (tba) play an octave lower than Group 2, with a minimum frequency of 38.19 Hz. Some instruments, such as the alto saxophone (as), trumpet (tp), flugelhorn (fh), and clarinet (cl), share a similar pitch distribution, each with 20 recorded tracks. Others, like the English horn (eh) and French horn (fho), have a more restricted range with 10 or 11 tracks, respectively (see Table 8.1).

8.3 Baseline Experiments

In this section, we present baseline experiments that completely eliminate cross-talk, as all instruments in the ChoraleBricks dataset were recorded in isolation. These experiments serve as reference points and establish an upper bound for evaluating the impact of cross-talk on pitch estimation in Section 8.4. We first introduce the pitch estimation algorithms (Subsection 8.3.1), followed by the evaluation metrics

(Subsection 8.3.2), and then analyze pitch accuracy for different instruments (Subsection 8.3.3) and MIDI pitches (Subsection 8.3.4).

8.3.1 Pitch Estimators

For our experiments, we use four widely adopted pitch estimation algorithms, each with distinct characteristics. First, `YIN` [35], an autocorrelation-based method, is known for its simplicity and speed but is prone to octave errors. Second, `SWIPE` [21] leverages a sawtooth waveform model to match spectral templates, reducing subharmonic errors by emphasizing fundamental and prime harmonics. This approach can be adapted for real-time applications with modifications [86]. Third, `PYIN` [82], an extension of `YIN`, incorporates a hidden Markov model to improve pitch estimates by considering temporal context, though it is not suited for real-time use. Lastly, `CREPE` [66], a DL-based method using a convolutional neural network (CNN), achieves state-of-the-art accuracy but is computationally intensive.

For `YIN` and `PYIN`, we use the implementations from `librosa`[23] [83]. For `SWIPE`, we rely on the `libf0` Python package [111]. For `CREPE`, we use the official implementation available on GitHub.[24] This version of `CREPE` is trained on more data than reported in the original paper and includes an improved argmax-local weighted averaging formula, enhancing accuracy compared to the original publication.

In our experiments, we use a sample rate of 44100 Hz, a hop size of 512 ($\approx$ 11.61 ms), and a window size of 4096 ($\approx$ 92.88 ms). For `CREPE`, we maintain the default step size of 10 ms (hop size of 441) for better accuracy. We do not apply the `CREPE` Viterbi post-processing for temporal smoothing as it is not part of the original model paper. The frequency range is set between note C1 ($\approx$ 32.70 Hz) and note A6 (1760 Hz) to cover our dataset's full range (see Figure 8.2). The pitch resolution is set to 10 cents for `SWIPE` and `PYIN`. Implementations of `YIN` and `CREPE` lack this parameter, but their post-processing methods, such as parabolic interpolation and local averaging, result in non-quantized frequency resolution.

8.3.2 Evaluation Metrics

In our experiments, we use evaluation metrics from the `mir_eval` Python package [107], focusing on RPA [104], which measures the percentage of melody frames where the estimated frequency matches the reference within a specified cent tolerance. To assess its impact on pitch estimation accuracy, we evaluate three tolerances: 50 cents (half a semitone), 25 cents, and 10 cents. While 50 cents is commonly used in pitch estimation evaluation, it may be too broad for applications like intonation analysis. Smaller tolerances, such as 25 and 10 cents, impose stricter requirements and pose greater challenges for pitch estimation algorithms. We do not include VR in this study, as our primary focus is on pitch accuracy in cross-talk scenarios.

[23] We specifically used commit `ebd878f`, which includes recent updates and bug fixes for `YIN` and `PYIN`.
[24] `https://github.com/marl/crepe`

RPA		CREPE			PYIN			YIN			SWIPE		
Cent Tol.		10	25	50	10	25	50	10	25	50	10	25	50
Woodwind	fl	0.907	0.971	0.979	0.938	0.971	0.984	0.939	0.965	0.976	0.776	0.953	0.966
	ob	0.969	0.989	0.993	0.954	0.975	0.983	0.952	0.969	0.976	0.962	0.986	0.993
	as	0.946	0.974	0.986	0.938	0.975	0.990	0.939	0.974	0.988	0.900	0.967	0.985
	cl	0.963	0.987	0.993	0.943	0.970	0.979	0.945	0.970	0.979	0.953	0.985	0.992
	eh	0.969	0.990	0.993	0.950	0.979	0.986	0.942	0.967	0.974	0.966	0.990	0.995
	bs	0.945	0.979	0.987	0.939	0.975	0.989	0.938	0.972	0.985	0.932	0.977	0.990
	bcl	0.897	0.946	0.973	0.946	0.968	0.977	0.950	0.970	0.980	0.958	0.981	0.988
	∅	0.942	0.977	0.986	0.944	0.973	0.984	0.943	0.970	0.980	0.921	0.977	0.987
Brass	tp	0.964	0.987	0.994	0.933	0.971	0.981	0.937	0.970	0.981	0.943	0.985	0.995
	fh	0.959	0.984	0.993	0.929	0.969	0.982	0.933	0.968	0.981	0.910	0.983	0.993
	fho	0.940	0.978	0.990	0.924	0.971	0.988	0.923	0.966	0.982	0.912	0.983	0.995
	bar	0.895	0.933	0.947	0.873	0.922	0.938	0.876	0.918	0.933	0.817	0.931	0.952
	tb	0.917	0.967	0.975	0.920	0.974	0.983	0.926	0.973	0.982	0.897	0.962	0.972
	tba	0.700	0.867	0.911	0.719	0.877	0.940	0.716	0.876	0.937	0.713	0.878	0.945
	∅	0.896	0.953	0.968	0.883	0.947	0.969	0.885	0.945	0.966	0.865	0.954	0.975
∅ **Overall**		0.928	0.967	0.979	0.918	0.961	0.976	0.919	0.959	0.973	0.898	0.967	0.982

Table 8.2: Mean RPA values for different estimators (CREPE, PYIN, YIN, SWIPE) using different tolerance values in cents (10, 25, 50) for woodwind and brass instruments, sorted by median frequencies.

8.3.3 Analysis Across Instruments

In Table 8.2, we present an overview of the mean RPA values for different pitch estimators, using various evaluation tolerances, shown for individual instruments. Additionally, we provide the average RPA values for the two subsets of instrument types, brass and woodwind, along with the overall average for the entire dataset.

Analyzing the overall RPA at a tolerance of 50 cents, all estimators perform comparably well, with SWIPE achieving the highest RPA value of 0.982. When the tolerance is reduced to 25 cents, accuracy declines slightly, with RPA values ranging from 0.959 for YIN to 0.967 for SWIPE and CREPE. At a lower tolerance of 10 cents, accuracy decreases more noticeably, with values spanning from 0.898 for SWIPE to 0.928 for CREPE.

In the following, we examine the RPA values with a 10-cent tolerance to highlight key differences and challenges among pitch estimation algorithms and instruments. On average, woodwind instruments are detected more accurately than brass instruments. For woodwinds, PYIN achieves the highest RPA at 0.944, while for brass instruments, CREPE leads with an RPA of 0.896. The tuba (tba), the most challenging instrument in the dataset, performs below its group's average and has its best RPA at 0.719 with PYIN, whereas CREPE yields 0.700. Although the flute (fl) is the highest-pitched instrument in the dataset, it shows weaker results: from 0.776 with SWIPE to 0.939 with YIN, possibly due to its turbulent noise or breathiness [79].

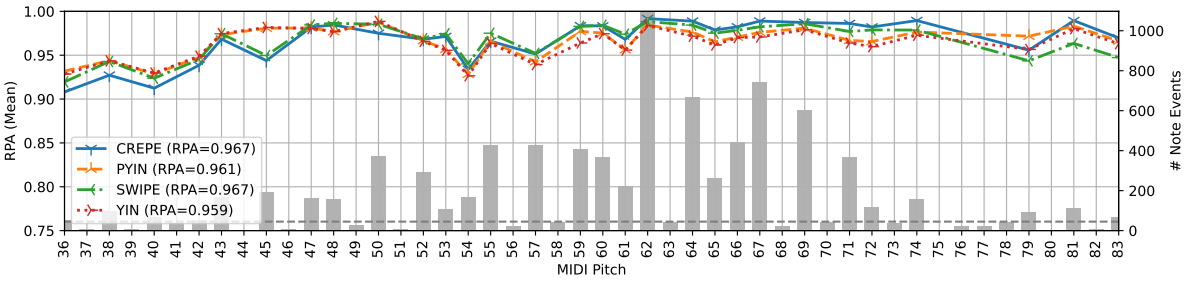


Figure 8.3: Pitch-dependent evaluation of pitch estimation methods on the `ChoraleBricks` dataset, displaying mean RPA values for estimators with a 25-cent tolerance. Gray bars show the number of note events per MIDI pitch from annotations. RPA data points are plotted only for notes with more than 45 events (gray dashed line).

In this analysis, `CREPE` generally emerges as the top-performing method. However, traditional methods like `YIN` or `SWIPE` sometimes slightly outperform `CREPE`, such as with the tuba (`tba`) or even exceed its performance, as seen with the bass clarinet (`bcl`). This may be because certain instruments are not part of `CREPE`'s training set, leading to less effective generalization. Incorporating wind instruments into `CREPE`'s training set could enhance its effectiveness and is a potential direction for future research.

While 10-cent tolerances are desirable for our application, they can lead to problems for three key reasons. First, many methods use interpolation techniques like parabolic interpolation or local averaging to achieve the desired output resolution. Second, small implementation details have a larger impact at a 10-cent error tolerance. Third, reference annotations and methods can introduce bias at the 10-cent precision level. Therefore, we will use a 25-cent tolerance for our analyses moving forward, which makes these issues less significant. Internal experiments also showed similar trends between the 10-cent and 25-cent evaluation tolerances.

8.3.4 Analysis Across Musical Instrument Digital Interface (MIDI) Pitches

Figure 8.3 shows the pitch-dependent evaluation results of four pitch estimation methods on the `ChoraleBricks` dataset. The figure plots the mean RPA values with a 25-cent tolerance against MIDI pitch values. We also include the number of note events for each MIDI pitch, derived from the F_0 annotations. RPA data points are only plotted for MIDI pitches with more than 45 note events, as pitches with fewer events may lead to less reliable RPA values due to insufficient data.

All pitch estimators perform well at the 25-cent tolerance, with mean RPA values mostly exceeding 0.95 and none below 0.9. The pitch curves generally remain flat, although a slight RPA drop is observed for all estimators at MIDI pitches 54 and 57. Below MIDI pitch 43, RPA tends to decrease for all estimators, with `CREPE` showing the lowest accuracy; however, this range exclusively includes notes played by the tuba (`tba`). Above MIDI pitch 74, `SWIPE` shows lower RPA values compared to other methods. This higher range features only a single instrument, the flute (`fl`). The higher and lower MIDI pitches are not supported by as many note events; therefore, they should be interpreted with caution. Additionally, as

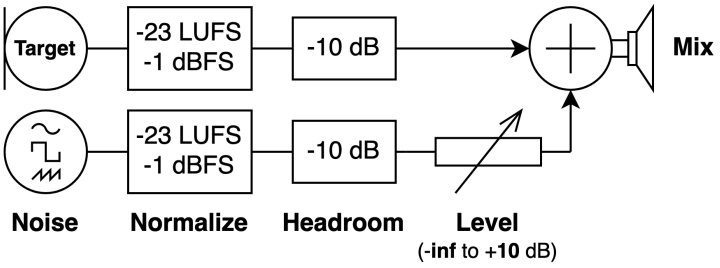


Figure 8.4: Flowchart for mixing instrument tracks with, e.g., Gaussian noise as interferer. See text for details.

discussed in Subsection 8.3.3, both the flute and tuba have been identified as among the most challenging instruments in this dataset.

Using a 25-cent evaluation tolerance, `CREPE` and `SWIPE` both show the best results with an RPA of 0.967 each. However, this analysis indicates that the RPA of estimators can vary depending on the pitch being analyzed. Specifically, we observe that `CREPE` performs worse for lower pitches, while `SWIPE` struggles more with higher pitches.

8.4 Cross-Talk Experiments

In this section, we simulate various cross-talk scenarios with increasing complexity: noise interference (Subsection 8.4.1), single-instrument interference (Subsection 8.4.2), and multi-instrument interference (Subsection 8.4.3).

In all three scenarios, we employ a similar signal processing flow to mix target instrument tracks with interfering signals, as illustrated in Figure 8.4. To clarify the general concept, we use the first scenario as an example (see Subsection 8.4.1). First, the target instrument is normalized to -23 LUFS (Loudness Units Full Scale), and a limiter is set at -1 dBFS (decibels relative to full scale) to prevent potential distortion from audio peaks. This setup complies with the *EBU R 128* broadcast standard, ensuring consistent loudness levels across different instrument recordings.

The interferer, whether it is noise, a single instrument, or multiple instruments, is normalized similarly to the target. The level of the interferer can be adjusted from $-\infty$ (meaning no interference) to +10 dB (where the interferer is even louder than the target). This adjustment allows for SNR levels ranging from ∞ to -10 dB in increments of 5 dB. To prevent audio clipping in the final mix due to the additional 10 dB boost from the interferer, we apply a headroom of -10 dB to both the target and interferer. This ensures that the final mix consistently remains below 0 dBFS for all SNR variations. An SNR level of 0 dB results in equal loudness for both the target and interferer.

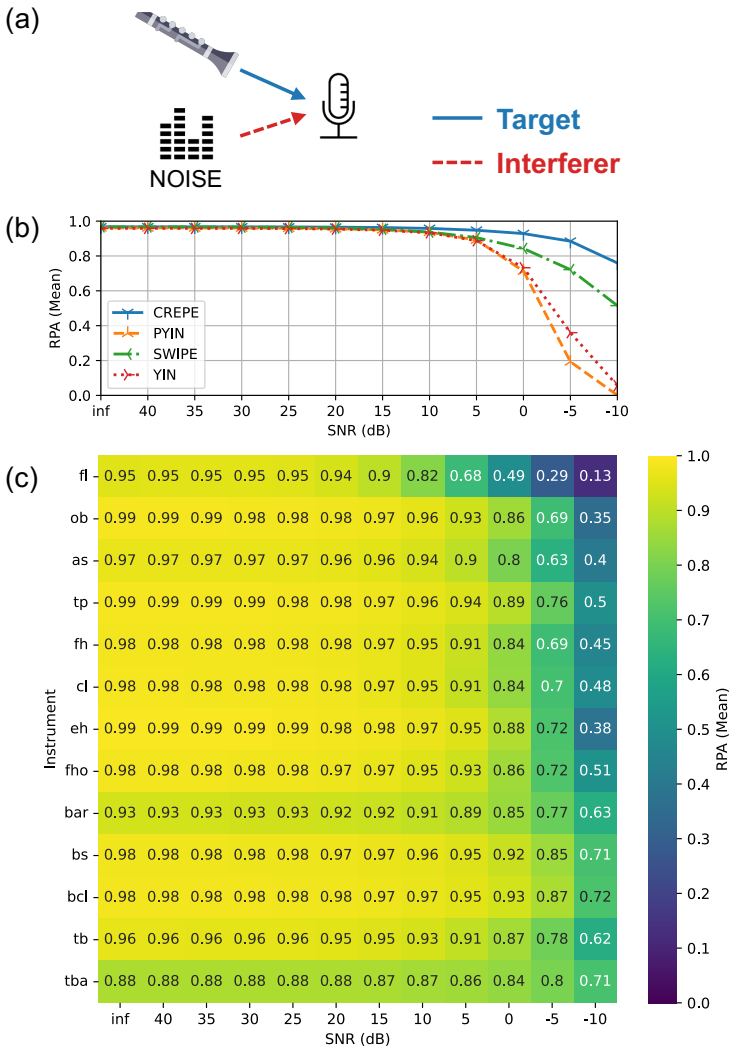


Figure 8.5: **(a)** First cross-talk experiment with Gaussian noise. **(b)** RPA vs. SNR for various pitch estimators at a 25-cent tolerance. **(c)** RPA vs. SNR for individual instruments using `SWIPE`.

8.4.1 Gaussian Noise Interference

For the first experiment, we add Gaussian noise at various SNR levels to all tracks in the `ChoraleBricks` dataset, as shown in Figure 8.5a. Figure 8.5b presents the mean RPA for different SNR levels and for several pitch estimators. All estimators yield similar results at SNR levels of 15 dB and higher, where accuracy plateaus and further noise reduction is unlikely to enhance RPA results. Below 15 dB, the estimators substantially diverge. For example, `CREPE` performs best with an RPA of 0.929 at 0 dB SNR, decreasing to 0.760 at -10 dB SNR. `SWIPE` follows with an RPA of 0.843 at 0 dB SNR, dropping to 0.515 at -10 dB SNR. `YIN` and `PYIN` show similar values of 0.732 and 0.713 at 0 dB but experience a significant drop in RPA at lower SNR levels.

Figure 8.5c illustrates the noise robustness of individual instruments with a 25-cent tolerance, using `SWIPE` as an example. At an SNR of 15 dB, all instruments achieve RPA values of 0.9 or higher, except for the tuba (`tba`), which remains below this value (see Table 8.2). The bass clarinet (`bcl`) exhibits the highest

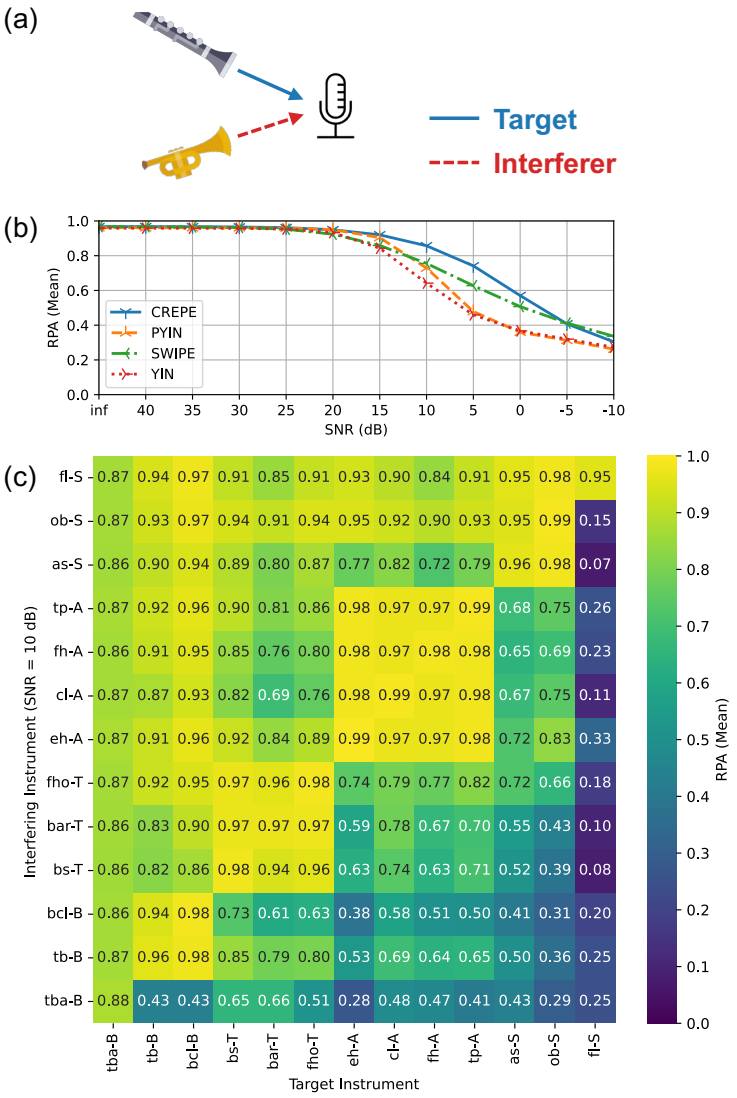


Figure 8.6: **(a)** Second cross-talk experiment with single-instrument interference. **(b)** RPA vs. SNR for different pitch estimators with a 25-cent tolerance. **(c)** Instrument and part combinations at 10 dB SNR using `SWIPE`.

robustness, maintaining an RPA of 0.72 at -10 dB SNR, closely followed by the tuba (`tba`) and baritone saxophone (`bs`). In contrast, the oboe (`ob`) and English horn (`eh`) are more sensitive to noise, with RPA dropping to 0.35 and 0.38, respectively, at -10 dB SNR. The flute (`fl`) is most affected, with its RPA decreasing sharply from 0.49 at 0 dB to 0.13 at -10 dB.

8.4.2 Single-Instrument Interference

In our second experiment, we introduce an additional instrument to interfere with the target, as shown in Figure 8.6a. We calculate various combinations of target and interfering instruments for different SNR levels, similar to the noise experiment in Subsection 8.4.1. For the following discussion, we use a notation where an instrument is appended with its part; for example, `fl-S` indicates a flute playing the soprano part, and `fho-T` denotes a French horn playing the tenor part. For each choral piece, we consider all possible

pairs of available instrument tracks, from tba-B to fl-S. We also allow mixing instruments of the same part, like two soprano instruments playing in unison (fl-S, ob-S), and scenarios where a target instrument track is mixed with its own recording, such as (cl-A, cl-A), which serves as a baseline for no interference. This method generates a total of 44,916 musically meaningful pairs of instrument tracks with varying configurations of target instruments, interfering instruments, and SNR levels.

In Figure 8.6b, we present the aggregated results of RPA versus SNR for different pitch estimators at a 25-cent tolerance. Estimating the pitch of a target instrument mixed with another interfering instrument is notably more challenging than in the previous noise experiment. At an SNR of 25 dB or lower, all estimators exhibit a substantial decline in accuracy. Even CREPE, which performed well with the noise interferer (RPA of 0.929 at 0 dB), drops to an RPA of 0.571 at 0 dB for the single-instrument interferer. SWIPE, YIN, and PYIN achieve RPA values of 0.507, 0.367, and 0.359 at 0 dB, respectively.

To better understand the conditions under which an estimator prefers one instrument over another, we select a representative set of instruments playing different parts (SATB) and analyze their mixes in terms of RPA, as shown in Figure 8.6c. In the matrix, each column represents a target instrument, while each row shows the interfering instrument mixed with the target at an SNR of 10 dB. We use the SWIPE algorithm as our example estimator. Instruments are sorted by their median frequency (see Figure 8.2). We ensure an equal balance for all instruments, with each part represented at least three times.

The main diagonal of the matrix, running from the bottom left to the top right, contains pairs where the target and interfering instrument recordings are identical, such as (tba-B, tba-B). These elements have no interference and serve as reference points. Along this diagonal, clusters of instrument parts are observed. The most prominent is the alto group spanning from eh-A to tp-A. Mixing instruments of the same parts means all instruments play the same notes in unison, likely resulting in high RPA values. Similar clusters appear for soprano, tenor, and bass, but do not include the tuba (tba) and flute (fl), which play an octave lower or higher, respectively. Below the diagonal towards the lower right corner of the matrix, RPA values decline drastically. This indicates that high-pitched target instruments show reduced RPA when mixed with low-pitched interferers. Conversely, as indicated by the upper left corner, low-pitched targets are less affected by high-pitched interferers.

When analyzing various target instrument columns, the flute, which has the highest pitch in our dataset, shows the lowest RPA values and achieves a maximum RPA of only 0.33 when paired with other instruments. In contrast, some lower-pitched instruments, such as the bass clarinet (bcl), often achieve high RPA values above 0.9 in most combinations. An exception occurs when the bass clarinet is mixed with the tuba (tba), which has an even lower pitch; here, the RPA value drops significantly to 0.43.

To explore the impact of low-pitched interferers on high-pitched target instruments, we examine a specific case where the alto saxophone (as-S) is the target, and the bass clarinet (bcl-B) is the interferer. According to Figure 8.6c, this combination yields an average RPA value of 0.41 across all songs in the dataset. In Figure 8.7, we analyze a single song (“Auf, auf, mein Herz, mit Freuden” by Crüger), showing waveforms

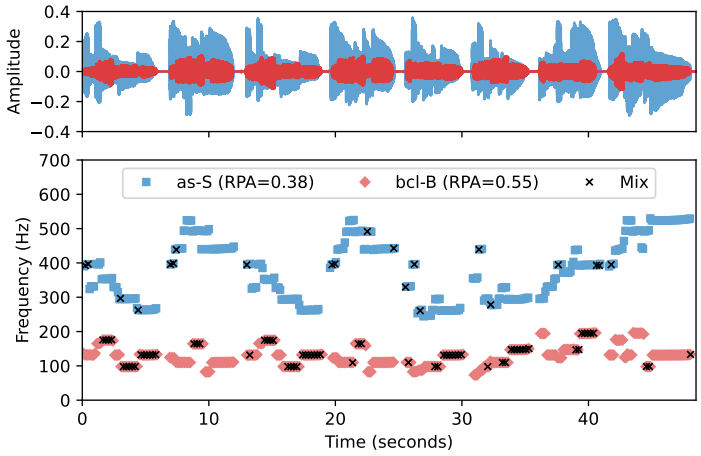


Figure 8.7: Waveforms and frequency trajectories for "Auf, auf, mein Herz, mit Freuden" by Crüger are shown for alto saxophone (`as-S`, blue) and bass clarinet (`bcl-B`, red) at 10 dB SNR. Frequencies are estimated with the `SWIPE` algorithm and evaluated with a 25-cent tolerance. The estimated trajectory for the mix is in black.

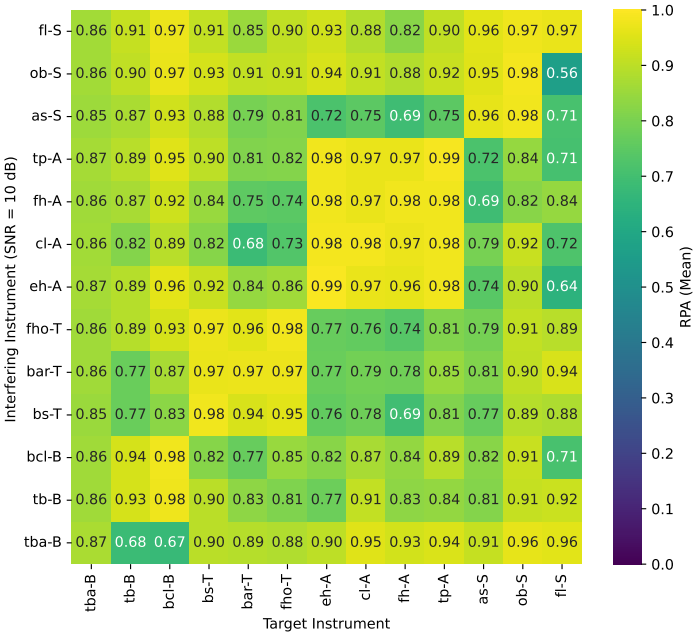


Figure 8.8: RPA vs. SNR for individual instruments, high-pass filtered at median frequencies, using `SWIPE` with a 25-cent tolerance.

and frequency trajectories for these instruments. Despite a 10 dB level drop for the bass clarinet (`bcl`) compared to the alto saxophone (`as`), the alto saxophone achieves an RPA value of 0.38. When switching perspectives and evaluating the mix with the reference annotation of the bass clarinet (`bcl`), it reaches an RPA value of 0.55. Together, these RPA values total 0.93, indicating that pitch estimation is almost always attributed to one of these instruments. This is illustrated in Figure 8.7, where the estimated trajectory for the mix (colored in black) consistently aligns with one of the two instrumental trajectories.

This example highlights a significant issue with cross-talk in pitch estimation: F_0 algorithms, like `SWIPE`, are often designed to estimate the lowest pitch in a signal, which becomes challenging when mixed with another instrument playing much lower notes. A straightforward solution may be to use a high-pass filter tailored to the frequency range of the target instrument, as demonstrated in Figure 8.8. For each mix, we apply a high-pass filter with a cut-off frequency at the median target instrument frequency (see Figure 8.2)

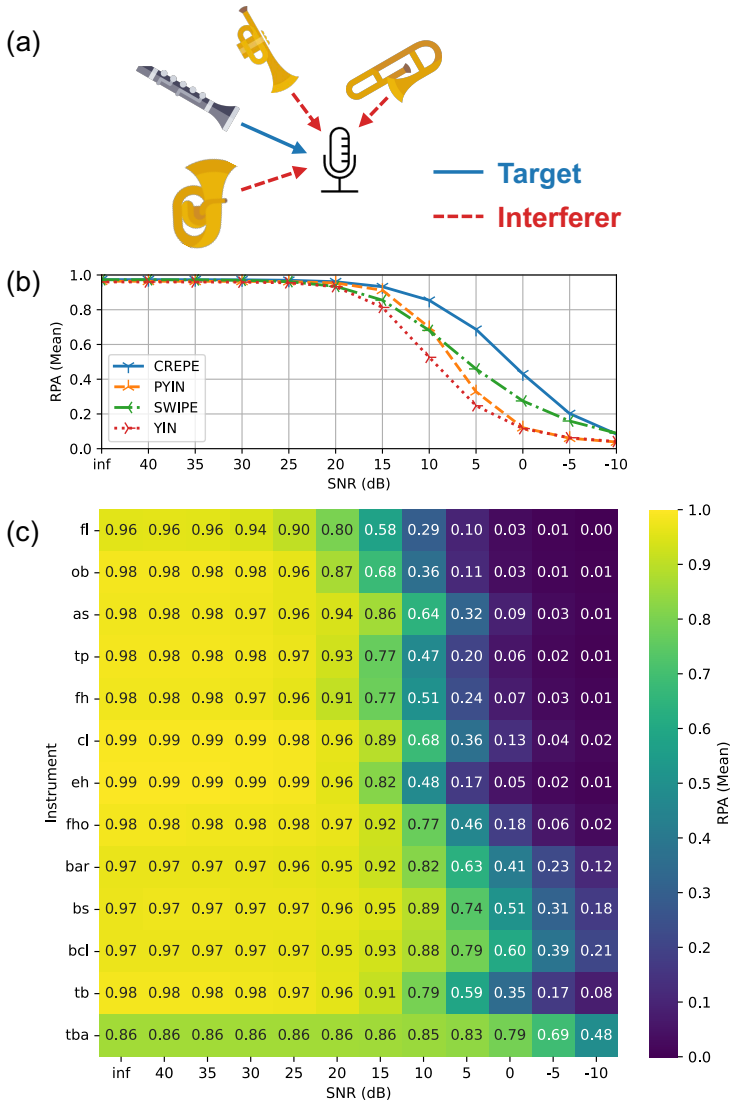


Figure 8.9: **(a)** Third cross-talk experiment with multi-instrument interference. **(b)** RPA vs. SNR for various pitch estimators at 25-cent tolerance. **(c)** RPA vs. SNR for individual instruments using `SWIPE`.

and with a slope of 24 dB per octave. For example, the mix with the oboe (ob) target is high-passed at 441.01 Hz, while the mix with the trombone (tb) target is cut at 148.37 Hz. This significantly reduces the impact of interfering instruments with lower note content compared to the unfiltered experiment (compare Figure 8.8 with Figure 8.6c). This approach could be an interesting direction for future research and support real-time intonation monitoring systems.

8.4.3 Multi-Instrument Interference

In our final experiment, we extend from single-instrument to multi-instrument interference, as shown in Figure 8.9, simulating typical ensemble performances where instruments are placed close together. We focus on quartets, where each of the four instruments plays a distinct SATB part. Each instrument is analyzed individually as the target while being mixed with a three-instrument interferer at various SNR

levels. An SNR of 0 dB indicates equal loudness between the target and interferer, see Section 4 for further details. We limit our experiment to a single piece, selecting the chorale "Auf, auf, mein Herz, mit Freuden" by Crüger, which has the most available instrument tracks in `ChoraleBricks`. We generate all possible quartet ensembles, ensuring that each instrument plays a distinct `SATB` part while excluding configurations like `SABB` or `BBBB`. This results in 36,000 audio files covering various target instruments, multi-instrument interferers, and SNR-level configurations.

In Figure 8.9b, we present the RPA values for various SNR levels, aggregated over all mixes for each pitch estimation algorithm and evaluated with a 25-cent tolerance. Compared to the previous experiment with a single interfering instrument, one can observe similar trends for high SNR levels down to 25 dB, where estimators start to diverge. At 15 dB, all estimators maintain an RPA above 0.8. However, for lower SNR levels, the RPA significantly drops, with all estimators falling below 0.5 RPA at an SNR of 0 dB or lower. This suggests that multi-instrument interference with distinct `SATB` parts poses a greater challenge for pitch estimation than single-instrument interference where some instrument combinations play in unison.

Figure 8.9c illustrates the RPA versus SNR for individual target instruments mixed with a multi-instrument interferer. Unlike the noise experiment in Subsection 8.4.1, where most instruments achieved an RPA over 0.9 at 5 dB SNR, achieving similar results in a multi-instrument setting requires a much higher SNR of 20 dB or more. This highlights the increased difficulty for estimators dealing with multi-instrument interference due to increased tonal complexity. Especially high-pitched instruments experience a more rapid decline in RPA with decreasing SNR levels, as they are typically mixed with lower-pitched instruments.

8.5 Conclusions

In this study, we examined pitch estimation in wind instrument cross-talk scenarios using the `ChoraleBricks` dataset. Our results demonstrate the significant impact of cross-talk, particularly at lower SNR levels. We found that `YIN` and `SWIPE` offer a strong balance between robustness and real-time capability, while `CREPE` provides higher accuracy at the cost of increased computational complexity. A key insight from our experiments is that lower-pitched instruments tend to dominate mixed signals, leading to biased pitch estimates. By applying frequency filtering and maintaining an SNR above 15 dB, we can mitigate these effects and improve pitch estimation accuracy. For future work, we aim to improve pitch estimation models through instrument-specific adaptations, such as enhanced filtering techniques or integrating instrument-specific templates in `SWIPE`. These adaptations may be complemented by lightweight machine learning techniques trained on data with cross-talk. We also intend to evaluate how cross-talk affects estimation accuracy in real-world recording environments. Finally, we will refine real-time intonation monitoring tools to support practical applications in ensemble performance and music education. In particular, the insights gained in this chapter prove valuable for the development of the real-time intonation monitoring system described in Section 9.1, which is one of the interactive pitch estimation applications presented in Chapter 9.

9 Interactive Pitch Estimation Applications

In this chapter, we present two interactive pitch estimation applications and a user study, all based on the real-time pitch estimation algorithms introduced in Chapter 6, specifically the `RT-SWIPE` algorithm refined in Chapter 7. The first application is a multi-user interface for real-time intonation monitoring in music ensembles (Section 9.1), which incorporates insights from the previous chapter on pitch estimation cross-talk (Chapter 8), with a particular focus on wind instruments. The second application is a prototype music game for singing (Section 9.2), enabling players to interact with a game world using their voices. Building on this prototype, educational gaming levels were designed to support singing practice, and a user experience study was conducted to explore the gamified singing concept in action (Section 9.3).

9.1 A Multi-User Interface for Real-Time Intonation Monitoring in Music Ensembles

This section is based on [90]. Peter Meier is the main contributor to this article. In collaboration with Stefan Balke and his supervisor Meinard Müller, he developed the ideas, designed the application, planned the experiments, and wrote the paper. Peter Meier also developed and implemented all approaches and conducted the user experiment. Stefan Balke contributed to the design and execution of the user experiment.

Intonation is a crucial aspect of musical ensemble performance. Yet maintaining accurate tuning among multiple musicians remains a persistent challenge, particularly in variable rehearsal and live performance settings. While intonation monitoring systems have been explored previously, most focus on single-channel processing, lack real-time capabilities, or do not support ensemble-specific functionality. In this work, we explore the possibilities of a real-time, multi-user intonation monitoring system specifically developed for music ensembles. It provides immediate, role-specific feedback to both musicians and conductors, supporting collaborative rehearsal practices and ensemble tuning processes. We conducted an initial experiment in a rehearsal setting with four musicians and a conductor, focusing on a qualitative evaluation of usability, interpretability, and musical relevance. The preliminary findings indicate that the system offers clear and practical feedback, promotes awareness of intonation, and has the potential to enhance interaction

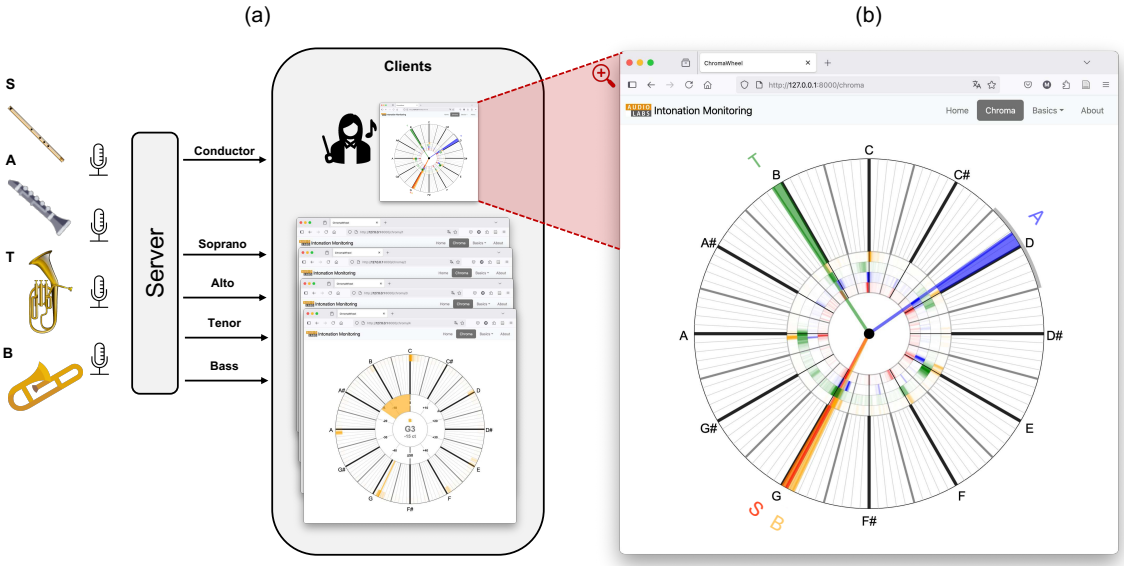


Figure 9.1: Overview of the real-time intonation monitoring system. **(a)** System architecture: Microphones capture audio from each musician, connecting to a central server for real-time pitch analysis and web-based visual feedback. **(b)** Conductor view: Main visualization for intonation monitoring, showing pitch class, cent deviation, and tuning stability for each musician.

and facilitate communication within the ensemble. These findings underline the potential of connected, real-time, multi-user interfaces to support ensemble musicianship and guide future developments in educational music technology.

The remainder of this section is structured as follows: Subsection 9.1.1 introduces the motivation, background, and related work. Subsection 9.1.2 details the intonation monitoring system, covering the pitch estimation algorithm and user interfaces. Subsection 9.1.3 presents the user experiment setup, procedure, and results. Finally, Subsection 9.1.4 discusses the system's technical challenges and musical impact, along with future development directions. Additional materials and resources are available on a supplemental website.[25]

9.1.1 Introduction and Related Work

Motivation and Challenges In amateur wind music ensembles, addressing intonation is often one of the most time-consuming aspects of rehearsal and concert preparation. While musicians typically tune their instruments to a reference pitch at the start, this does not guarantee that all members remain in tune throughout the session. Each instrument and player has unique intonation tendencies, and some pitches are inherently more difficult to tune accurately [47]. Musicians must make continuous adjustments during rehearsals and performances, especially as the musical context changes and affects how notes should be intonated.

[25] https://www.audiolabs-erlangen.de/resources/MIR/2025-ICMI-Intonation

Intonation in Wind Ensembles Many wind instruments allow for continuous pitch adjustment through playing techniques, air support, and tuning slides. As a result, intonation in wind ensembles is not fixed to 12-tone equal temperament and often gravitates toward just intonation [124]. Musicians adjust their tuning to the harmonic context, such as lowering major thirds, to achieve more consonant intervals within chords. These micro-adjustments require heightened awareness and real-time adaptation, especially when the harmony or chord structure changes rapidly.

Related Work To help musicians with intonation, computer-aided feedback systems have been explored in various music teaching contexts. For example, in [100], auditory and visual feedback was used to guide violin students in improving their intonation. In [7], real-time feedback was based on combined sound and motion analysis to support the development of sound quality. Similarly, [103] focused on assessing sound quality, but relied solely on acoustic features. A web-based user interface for supporting choral singing practice was introduced in [110], and a software tool for multi-channel audio analysis of singing voices was presented in [76], offering pitch-related feedback for singers.

9.1.2 Intonation Monitoring System

In this section, we describe the technical components and design principles of our real-time intonation monitoring system.

System Overview Our system enables real-time intonation monitoring for music ensembles, combining multi-channel audio input, robust pitch estimation, and intuitive visual feedback for both musicians and conductors. As illustrated in Figure 9.1a, microphones capture each instrument and transmit audio to a central server, which performs pitch analysis and distributes results to client devices for visualization.

Definition of Intonation In this work, **intonation** refers to the accuracy of the fundamental frequency in relation to a selected tuning system—specifically, the 12-tone equal temperament based on a reference tuning (e.g., 442 Hz). However, our approach can be easily adapted to consider the harmonic structure of the input signal [119], accommodate different tunings [81, 124], or make relative measurements between musicians [134].

Pitch Estimation Pitch estimation is performed using a real-time implementation of the `SWIPE` algorithm [21, 86]. `SWIPE` estimates the fundamental frequency by comparing spectral templates to the input signal and selecting the best-matching template. Because these templates incorporate harmonic structures similar to those produced by wind instruments, the algorithm performs well in this context. Moreover, in previous work, it was found that `SWIPE` is not only computationally efficient but also robust in the presence of acoustic cross-talk, a common challenge in multi-track ensemble recordings [91]. These

properties make SWIPE especially well-suited for real-time, multi-channel pitch tracking in live ensemble settings. The estimated pitches are processed on the server and then distributed in real time to multiple client devices, such as mobile computers or tablets, where they are visualized through a web-based application.

User Interfaces Our intonation monitoring system provides two main views: a **single view** for individual musicians and a **conductor view** for ensemble-wide monitoring. Both use a circular layout, specifically a *chromatic circle* [93], which resembles the *circle of fifths*, a concept familiar to most musicians. The grid is divided into 12 segments for the chroma classes, arranged clockwise with C at the top, condensing pitch information across octaves. In the conductor view, intervals and chords form recognizable patterns, enabling quick visual identification. The circular arrangement also supports smooth transitions across notes and octaves, minimizing abrupt jumps in the visualization of estimated pitches.

Single View The single view (Figure 9.1a, right side) is designed to provide each musician with real-time feedback on their own intonation, functioning similarly to a standard tuning device but enhanced with a circular layout for improved musical context. The outer circle displays the current pitch class as a pointer, allowing musicians to quickly identify which note they are playing. Within the inner circle, cent deviations from -50 to $+50$ cents are visualized, referencing the 12-tone equal temperament system and helping users recognize subtle tuning differences. To further support musicians, the interface incorporates pitch histograms that reflect tuning stability over time, enabling users to observe trends and fluctuations in their intonation during sustained notes or passages. Triangle markers highlight key musical intervals, such as pure major and minor thirds at -14 and $+16$ cents, respectively, providing visual cues for context-sensitive tuning adjustments.

Conductor View Figure 9.1b shows the conductor view for ensemble-wide monitoring, visualizing each musician's current note and cent deviation as colored pointers and areas. In our example, the ensemble plays the notes G, B, and D, which form a G-major chord. The soprano (red) and bass (yellow) play the G; however, the bass is slightly lower than the soprano, as indicated by its position on the outer ring. The B is played by the tenor (green), also a few cents lower. It is important to recall that in just intonation, the major third should be played at -14 cents. Finally, the alto (blue) is playing a D, which is also too low. The inner rings display a histogram for every connected device in their respective colors. This setup enables the conductor to make informed judgments about ensemble intonation that go beyond the capabilities of traditional isolated tuning devices.

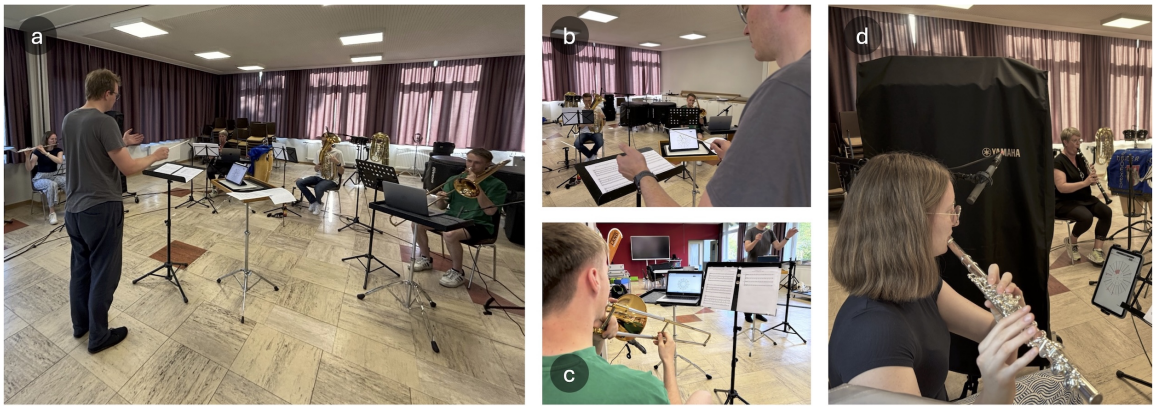


Figure 9.2: Impressions from the user experiment: **(a)** The ensemble seated in a semi-circle during rehearsal. **(b)** The conductor view with a multi-channel interface. **(c)** The trombone player's setup using a notebook. **(d)** The flute player using a tablet for the single view interface. All musicians agreed to make the photos available.

9.1.3 User Experiment and Evaluation

Experiment Setup The initial user experiment aimed to assess usability and inform future design of the intonation monitoring system in a real-world setting. It was conducted in the rehearsal room of the Weserberglandorchester (WBO) Bödexen, Germany, with four amateur musicians (flute, clarinet, baritone horn, trombone) and a conductor. The session, including discussion, lasted about two hours. As shown in Figure 9.2a, musicians were seated in a semicircle with approximately 2 meters between them. Each had a music stand and a display device (notebook or tablet) running the intonation monitoring interface, role-specific for musicians and a full overview for the conductor (Figure 9.2b–d). Four identical cardioid condenser microphones (Schoeps MK 4) were placed about 15 cm in front of each instrument.

Procedure The experiment session was organized into five sequential parts: instrument tuning, scale playing, chords, cadences, and chorales. Initially, all musicians tuned their instruments simultaneously using the intonation monitoring system, followed by individual pitch recordings and verification with the conductor's tuning device. Next, the ensemble played scales together, with the conductor indicating note changes; this was performed both with and without the monitoring system to compare experiences. The third part focused on B♭ major and B♭ minor chords in various inversions, with each musician taking turns playing the third and adjusting their intonation based on system feedback. In the fourth part, the group performed four different cadences, and finally, two chorales (`DR1` and `TE1`) from the ChoraleBricks dataset [1] were played. For each musical task, recordings were made both with and without the system, enabling direct comparison of intonation accuracy and ensemble interaction. After the musical session, all musicians completed a structured questionnaire covering three areas: personal and musical background, impressions of the intonation monitoring system, and open feedback.

Results Two participants were female and two were male, with ages between 19 and 50 years. All had been playing their main instrument for over ten years, practiced approximately two to three hours per week, regularly participated in at least one ensemble, and also played additional instruments. The impressions of the intonation monitoring system was assessed through statements rated on a 5-point Likert scale (from strongly disagree to strongly agree). Most participants reported a clear understanding of the information presented, and they generally found the interface to be intuitive. The display was not found to be distracting during playing, and participants noted an increased awareness of intonation while using the system. Ratings regarding the alignment between the display and their own auditory perception of intonation were neutral. Participants also expressed a neutral stance on using the system in future rehearsals, pointing to both interest and opportunities for further development.

User Feedback Overall, the participants considered the intonation monitoring system as a helpful tool, particularly appreciating features such as cent markers for thirds, the histogram visualization, and visual support for stabilizing pitch especially during sustained notes. They reported that the system enhanced their intonation awareness, revealing unexpected tendencies to play sharp or flat and helping them adjust pitch more accurately within chords. However, limitations were noted, especially with display instability during fast passages. Users suggested improvements including a refresh button, octave-specific histogram views, faster pitch display response, and overall stability enhancements. Although the conductor view was not specifically evaluated, initial impressions indicated that it was helpful for assessing ensemble stability and understanding the harmonic context, especially during long sustained notes. A more detailed discussion on the musical impact of the system is provided in the Subsection 9.1.4.

9.1.4 Conclusion and Future Work

Technical Challenges Our initial user experiment provided valuable insights into the real-time intonation monitoring system for music ensembles and highlighted technical and musical challenges for future iterations. A major issue was acoustic cross-talk, particularly with the flute, which is difficult to isolate from nearby instruments. The flute's smaller size and quieter sound are often overshadowed by larger, louder instruments like the baritone horn, complicating fundamental frequency estimation—a limitation noted in previous studies [91]. Acoustic room reverberation and ensemble spacing introduced further complexities. To improve signal isolation, future systems could combine better microphone selection, pre-processing techniques (e.g., low-pass filtering or source separation), and alternative input devices based on vibration sensors [61], similar to using larynx microphones in vocal applications [116].

Musical Impact Beyond technical aspects, the field study was highly rewarding. Musicians were patient, curious, and engaged, providing valuable feedback and appreciating the technology's novel perspective on their practice. The system increased their sensitivity to intonation and encouraged musical

reflection. Such tools can support not only tuning but also deeper engagement with ensemble playing. For conductors, the interface translates abstract instructions into concrete visual cues. For example, lowering the third in a major chord is often difficult for amateur musicians to internalize. Real-time feedback helps them intuitively understand and apply such adjustments, improving listening skills and intonation over time.

Future Directions Throughout the experiment, various ideas for future development emerged. Unlike default tuning devices, our system is a connected solution that enables advanced signal processing pipelines—for example, to address cross-talk and other ensemble-specific challenges discussed above. This connectivity allows for richer, context-aware feedback and collaborative features that go beyond isolated tuning support. Currently, only the conductor has access to the full ensemble's tuning data. However, musicians could benefit from context-aware feedback on their individual devices. This feedback might include which notes are being played by others, an estimate of the current chord, who is playing the root of that chord, their own harmonic role within the chord (e.g., major third), and a corresponding intonation target. Furthermore, if the ensemble drifts in pitch collectively, musicians could make informed adjustments rather than adhering rigidly to a fixed reference grid. Such contextual awareness could foster a more responsive and musically sensitive approach to ensemble intonation beyond what individual tuners offer.

9.2 Sing Your Way: A Music Game for Creative Vocal Interaction

> This section is based on [87]. The first author, Peter Meier, is the main contributor to this article. In collaboration with Simon Schwär and his supervisor Meinard Müller, he developed the ideas, designed the application, and wrote the paper. In addition, Peter Meier implemented all approaches and created the music game prototype. Gerhard Krump was the supervisor of the main author at the Technische Hochschule Deggendorf.

Music-based games are an important genre in the gaming community and have become increasingly popular with games like SingStar and Guitar Hero. These types of games are usually based on reactive game mechanics, where the player must hit a certain note at a certain time in order to score points. In this contribution, we present a game prototype that goes beyond purely music-reactive game mechanics and focuses more on the creative aspect of making music in games. In particular, we developed a jump-and-run game that can be controlled with a gaming controller but also uses the player's singing voice to interact with the game world. To this end, we estimate the pitch of a microphone signal in real time and use it as

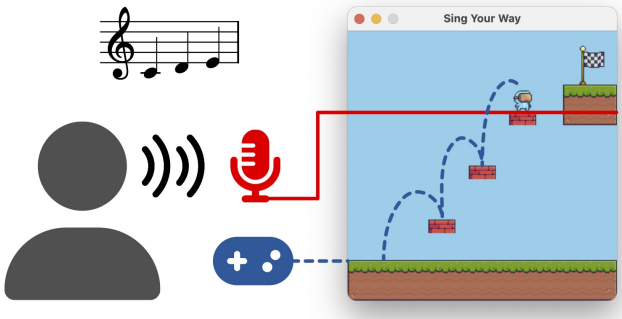


Figure 9.3: A demo level from our "Sing Your Way" music game prototype. It uses a combination of traditional gaming controller and singing voice input to add note blocks to the game and to reach the end of a level.

a creative input to the game. This input can be used to control parts of the game world, for instance by singing and adding stair-like elements that allow the player to overcome obstacles and reach the end of a game level. With our game prototype, we demonstrate how game designers can incorporate musical challenges into a well-known game environment while motivating musicians to creatively explore and practice their musical skills. The remainder of this section is structured as follows. In Subsection 9.2.1, we examine the game mechanics and interaction design of our music game prototype. Subsection 9.2.2 then explores the educational potential of singing and gamification, illustrating how the game can support both musical training and creative exploration.

9.2.1 Game Mechanics and Interaction

This work is part of a series on "Real-Time Signal Processing Algorithms for Interactive Music Analysis Applications" [84]. In previous work, we presented a music-reactive game with beat tracking, where the game world is generated in real time from ambient music (e.g., radio, live music) of the player [85] (see Section 5.2). In this chapter, we consider a different type of music game that uses pitch estimation as a control input. In developing this game, we want to encourage the creative use of melody and singing for players and game designers and concentrate on exploring and teaching real-time aspects of pitch estimation. In particular, we focus on two pitch estimation algorithms, YIN [35] and SWIPE [21], which we adapted from the Python library libf0 [111] and modified to work in real time for our gaming context.

We developed a music game prototype "Sing Your Way" in which players can interact with the game world by using their singing voices. Figure 9.4 shows what this might look like for a simple use case. In the first step, the player can sing a note. The pitch of this note is displayed as a red line in the game. The higher the note is sung, the higher the red pitch line is displayed. While singing a note, the player can interact with this red pitch line by jumping on it. By doing this, a fixed note block is created in the game for the player to stand on. The same step can be repeated several times at different pitches. As a result, players can add stair-like elements to the environment, using their voices to overcome obstacles, avoid enemies, or reach the end of a level.

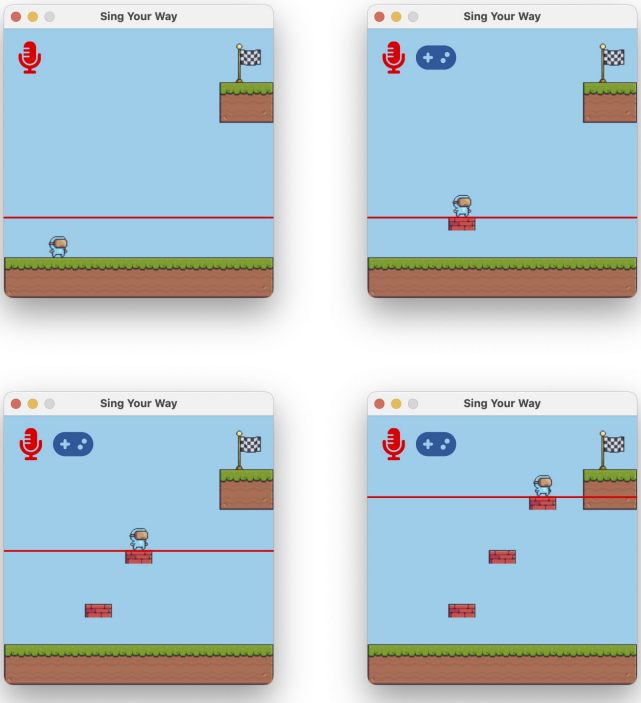


Figure 9.4: Gameplay sequence showing how to add note blocks and stairs to the game: Sung notes appear as a red line that can be jumped on. This creates a fixed note block on which the player can stand. The pitch of the notes corresponds to the vertical position of the note blocks.

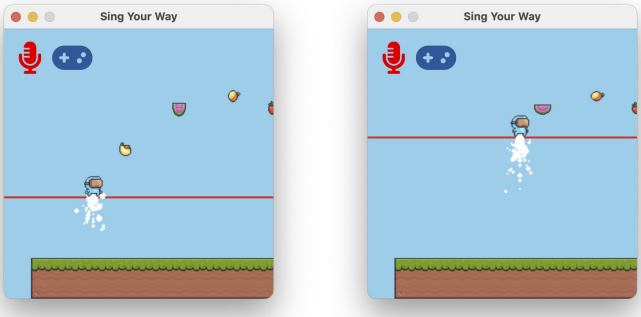


Figure 9.5: A gameplay sequence showing how to use continuous singing to make the character fly and collect items. In "flight mode", the vertical position of the character corresponds to the pitch of the notes. The horizontal position of the character is still controlled by a gaming controller.

A second game mode is shown in Figure 9.5. By pressing and holding a button on the gaming controller, the player can activate "flight mode". In this mode, the player's vertical position is controlled by the singing voice. The higher the player sings, the higher the player flies in the level, while horizontal movement is still controlled by the gaming controller. This expands the control options of the game and allows the player to reach certain areas of a level that would otherwise be unreachable by simply jumping and running.

9.2.2 Singing Education and Gamification

One of the main goals of our game prototype is to create a platform that connects scientists, level designers, and musicians, especially in an educational context. For this reason, we have integrated the ability to

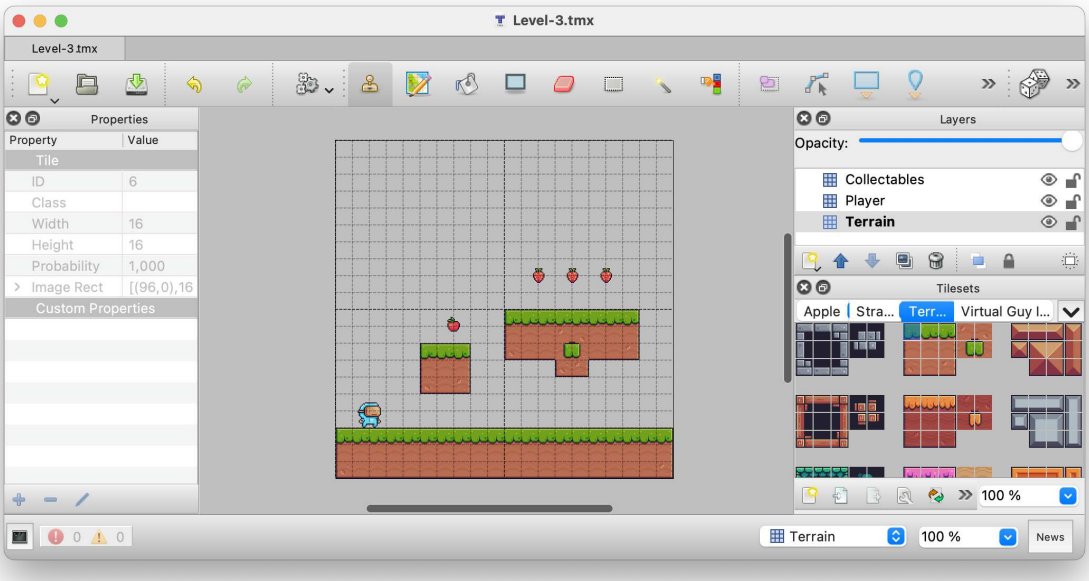


Figure 9.6: Simple and easy-to-use level design with the free and open source level editor "Tiled." It uses sets of simple graphical elements, called "tilesets", that can be placed on different layers of the map to create a playable level.

design levels using a map editor[26] and a tileset[27], as shown in Figure 9.6. Firstly, this allows for creative level design without any technical background in programming. Secondly, it gives researchers new and creative ways to test real-time algorithms in the context of a music game. Finally, this platform could also be interesting for musicians to creatively explore and practice their musical skills.

This could be done with different musical challenges to make practicing music more fun through gamification:

- **Melodies**: The player has to add blocks of notes to the game to make up a given melody, which can be loaded from a MIDI file, for example. If the correct sequence of notes is recognized, points are awarded or a level is successfully completed.
- **Musical scales**: The player is given a root note and must sing a musical scale (such as major, minor, etc.) to add note blocks to the game. These scales can be rewarded with points if the interval steps are correct or the general intonation of the scale is correct.
- **Intervals**: The player can be challenged to find various intervals for certain root notes in order to score points or achieve game goals. In this way, there can be different levels where the focus is on different intervals to make the training more enjoyable.

26 `https://www.mapeditor.org`
27 `https://pixelfrog-assets.itch.io/pixel-adventure-1`

Figure 9.7: Impressions from the user experiment at Dagstuhl Seminar 24302 *"Learning with Music Signals: Technology Meets Education"* (`https://www.dagstuhl.de/24302`): Participants are shown playing the game prototype and providing feedback in one-on-one discussions. The participants agreed to make the photos publicly available. All photos were taken by Sebastian Stober. The group photo is available at the Dagstuhl website.

9.3 Exploring Gamified Singing: A User Experiment at Dagstuhl

This section is related to Dagstuhl Seminar 24302 *"Learning with Music Signals: Technology Meets Education"*, specifically Section 3.13 of Dagstuhl Report 14(7):115 [95] (`https://www.dagstuhl.de/24302`). The first author, Peter Meier, is the main contributor to this article. In collaboration with Simon Schwär and his supervisor Meinard Müller, he developed the ideas, designed the user experiment, and wrote the paper. Additionally, Peter Meier implemented all approaches, designed the game levels, and conducted the user experiment.

In Section 9.2, we developed a prototype for an interactive music game called "Sing Your Way." The prototype resembles a jump-and-run game and uses a gaming controller as the input device. Additionally, it incorporates the player's singing voice to interact with the game world. For this, we estimate the pitch of a microphone signal in real time and use it to control parts of the game world. For example, by singing a note, a pitch line appears in the game that a player can jump on to add stair-like elements, allowing them to overcome obstacles and reach the end of a level. The game is deliberately kept simple, with a limited set of rules: Players can move left, move right, jump, and sing notes, where pitch determines the vertical position in the game world.

Within this simple gaming environment, we consider two specific use cases: singing education and data collection for MIR research. First, the prototype serves as a tool for singing education, offering a fun and motivating environment for practicing singing techniques. Players are challenged with different gaming levels that include various voice-related tasks, such as singing the correct notes, intervals, chords, scales, or melodies. Combining gamification with educational objectives helps players train their ear and improve their vocal control in an engaging and playful manner. Second, the game can also serve as a valuable tool for collecting data on vocal performance and singing style. By capturing detailed information about players' pitch accuracy, timing, and vocal range, educators and researchers can analyze trends and patterns in singing proficiency. This data can contribute to research in music education and assist in developing new, personalized methods for interactive vocal training and assessment. At the Dagstuhl Seminar, we presented a demonstrator of our game and invited feedback and discussions on the following topics: user experience, educational impact, technical improvements, potential collaborations, and data privacy. By actively playing our music game prototype, the Dagstuhl Seminar participants provided diverse perspectives and ideas on game level design for music education and MIR research.

The following subsections provide an overview of the designed game levels (see Subsection 9.3.1) and summarize the main findings from the user experience evaluation (see Subsection 9.3.2). Participants offered valuable feedback and insights while interacting with the game, which informed our analysis and future improvements. Additional gameplay videos are available on a dedicated project website.[28]

9.3.1 Game Levels

For the Dagstuhl seminar, we designed a series of prototype levels (Levels 0–5), each with specific goals and challenges. These levels are described in detail below.

Level 0: Introduction to Game Mechanics In Level 0, the player is introduced to the general game mechanics. The game window displays a jump-and-run style game where the character can be controlled by the player using a traditional gaming controller. The player can move left, right, and jump to overcome obstacles; however, the jump height is limited. To reach higher terrain, the player must utilize an additional game control mechanism. By using their singing voice to sing a note, a pitch line appears in the game. The vertical position of this pitch line represents the fundamental frequency (F_0) of the singing, with varying frequencies for higher or lower pitches.

The player can jump onto this pitch line to create a **note block**. This note block stores the pitch at the moment when the player lands on it from above. Jumping on the note block again reproduces the stored pitch, allowing players to make sounds within the game and potentially play notes or even entire melodies if multiple note blocks are involved. Furthermore, these note blocks serve as steps to help reach higher terrains and ultimately complete each level.

[28] https://www.audiolabs-erlangen.de/resources/MIR/2024-Dagstuhl-Music-Game

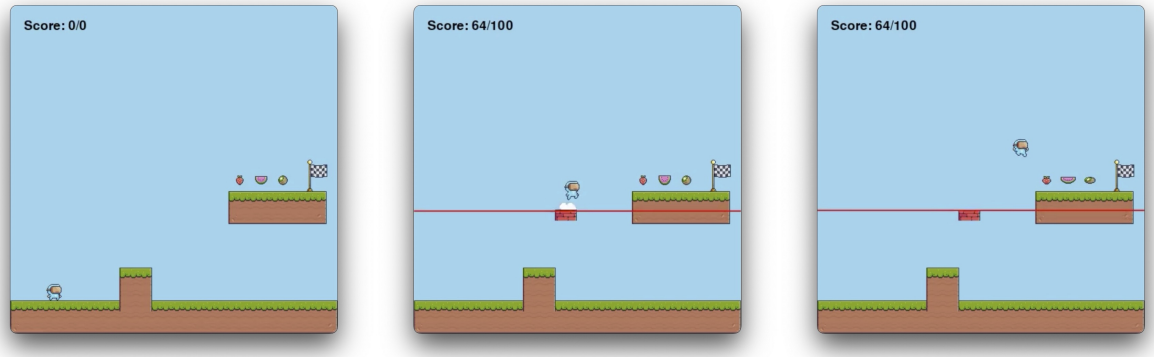


Figure 9.8: Level 0 introduces the core game mechanics, where players use their singing voice to create pitch-based note blocks. These blocks allow the player to jump to higher terrain, demonstrating the interaction between vocal pitch and gameplay.

Note blocks can be **deleted** by pressing the dedicated delete button while jumping on the blocks. This is useful for removing or correcting a misplaced note block to achieve better scores (more on that later) or to clear the path in a game level. Furthermore, this delete feature enables the player to creatively use note blocks to build something, such as a staircase or bridge to other platforms.

Another feature of note blocks is the ability to **mute** them by pressing the dedicated mute button while jumping on them. With this feature, the player can add soundless blocks into the game. This can be useful for challenging the player with certain tasks, such as creating a specific interval with note blocks and playing the notes back one after another. For such a task, the player would need to climb up or down a staircase without making a sound.

When the player reaches the high plateau in Level 0, fruits can be **collected** simply by moving into them. Like note blocks, collecting fruits also produces a sound by playing a short note that corresponds to their vertical position in the game. The difference with note blocks is that fruits disappear once they are collected. In this way, fruits not only serve as collectibles (potentially granting points) but also guide players to hear or sing certain pitches for a single time. This opens up interesting possibilities for different game level designs to challenge the player, which we will discuss in subsequent examples.

When starting the game, a **root note** can be set that matches the vertical position of the ground terrain in the game. This is useful for adjusting the pitch line to a comfortable singing height for each player, as there might be significant differences between male and female voices.

The **score** displayed in the upper left corner of the game level is calculated based on the stored pitch values (in cents) of all note blocks in the game. The closer a note block lands to a fixed half-note grid (in cents) based on the root note, the higher the score. For example, when a player creates a C3 note with 0 cents deviation, they are granted full points of 100. However, if the player is 20 cents off the grid, they receive only 80 points. In this way, players can train to hit pitches more accurately. The maximum possible score

Figure 9.9: Level 1 introduces a musical challenge where players collect fruits to hear reference pitches and then reproduce these pitches by singing and creating note blocks. This level helps players practice listening to and matching pitches while navigating the game environment.

Figure 9.10: Level 2 challenges players to master both whole and half step intervals by creating self-placed note blocks. Collectable fruits provide reference pitches, encouraging players to practice precise pitch matching while navigating the level without external aids like grass blocks.

for the current note blocks in the game is also given as a value separated by a slash behind the actual score, e.g., like 80/100. This way, players know they have earned 80 out of 100 possible points.

Level 1: Listening and Singing Notes In Level 1, the player is challenged to select each fruit and replace it with note blocks that have corresponding pitches. This allows the player to train their musical skills by listening to a given pitch (by collecting the fruit) and reproducing this pitch with their own singing voice (by singing a note, holding it, and jumping on it). The fruits are arranged so that the player ends up singing the first three notes of a major scale, with each note being a whole step interval apart. To help the player reach higher terrains, there are also fixed grass blocks that the player can use to stand on.

Level 2: Mastering Simple Intervals Similar to Level 1, in Level 2, the player has to collect fruits and replace them with note blocks to reach the end of the level. This time, there are not only whole step

Figure 9.11: Level 3 challenges players to sing a complete major scale while navigating the level. Collectable fruits provide reference pitches, helping players practice pitch accuracy and scale progression. The level requires precise control of both singing and character movement to succeed.

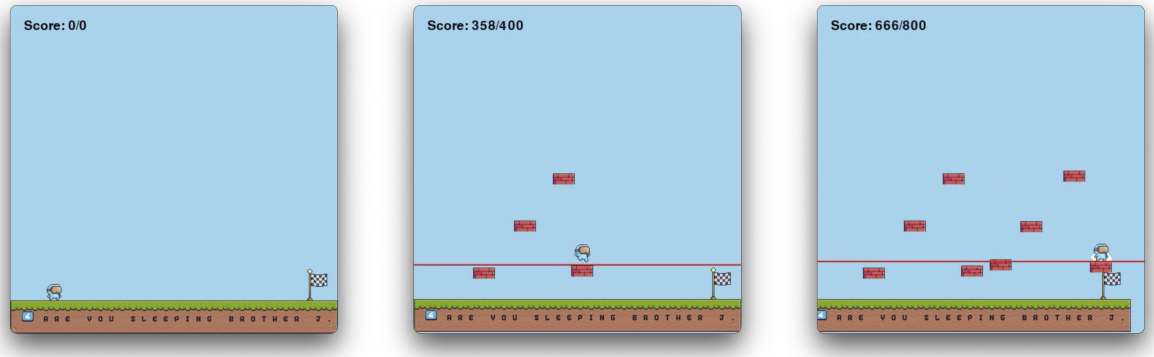


Figure 9.12: Level 4 challenges players to creatively build a melody by singing the children's song "Are You Sleeping, Brother John" and placing note blocks into the game. Players must use their singing voice to construct the melody while navigating the level, offering a unique blend of creativity and precision.

intervals but also half step intervals, as the fruits are given in such a way that they represent the first five notes of a major scale. However, there are no aiding grass blocks anymore, so the player must climb up using only the note blocks they created themselves.

Level 3: Major Scale Challenge With Level 3, we extend the challenge from Level 2 to encompass an entire major scale from bottom to top. In doing so, we end up with half-tone steps between the 3rd and 4th, as well as the 7th and 8th positions of the scale.

Singing eight notes in succession while jumping on note blocks without falling to the ground requires a high level of precision in controlling the player character with the game controller. This adds another layer of complexity to the game and becomes more difficult the longer the scale is to sing.

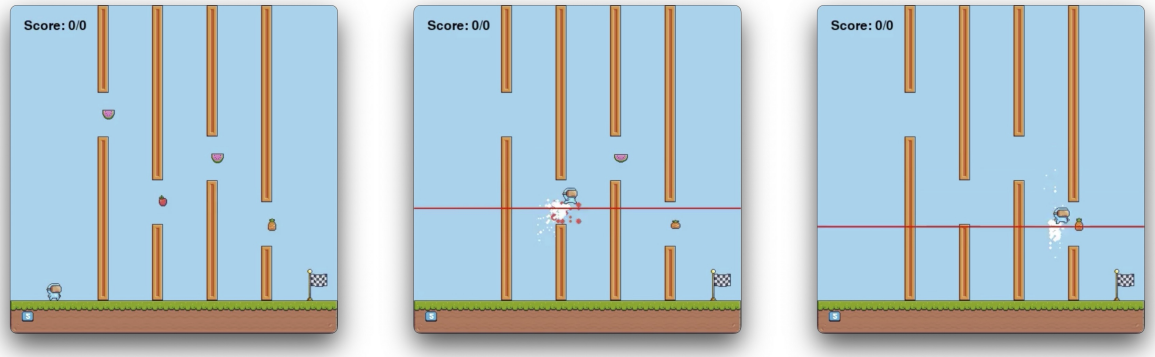


Figure 9.13: Level 5: With flying mode, players control their vertical position by singing pitches to navigate through the level. This game mechanic is inspired by *Flappy Bird*.

Level 4: Sing a Song In Level 4, we introduce a new game level idea by presenting the player with an empty level and one task: Sing the first few lines of the children's song *"Are You Sleeping, Brother John"* into the game. The note blocks must all fit within the initial game window. The order of the blocks is not important, but it should be possible to play back the melody by jumping on the blocks one by one.

As there are no reference tones in the form of a fruit or note block in the game, players can use a new guidance method: the **drone tone**. This tone represents the root note, which is set individually for each player at the startup of the game and can be activated or deactivated by pressing a dedicated button on the controller. This gives players great flexibility to creatively solve this task, but it also introduces interesting new challenges, such as solving it very quickly (speed run), achieving high accuracy in pitch, or completing the level with or without using the drone tone.

Level 5: Flappy (Bird) Beethoven Level 5 introduces an exciting new game mechanic: **flying mode**. By pressing and holding a dedicated button on the gaming controller, the character enters flying mode, and its vertical position is set to the pitch line. In this mode, players can fly through narrow pipes or gaps in the level, similar to games like *Flappy Bird*, but this time it is controlled by the player's singing voice.

The flying mode also offers another type of vocal skill that can be developed. Instead of singing stable, steady pitches one by one, players can smoothly slide between different pitches to change their vertical position in the game. This continuous change of pitch provides a fun and interesting way to train the voice. For example, the player could (a) try to navigate through the pipe course in one take with just one breath and (b) aim to avoid touching the pipes to reach the end of the level. While solving Level 5, the player might notice the famous four-note "short-short-short-long" motif from *Ludwig van Beethoven's Symphony No. 5* in C minor, Op. 67.

9.3.2 User Experience Evaluation

To evaluate the user experience, we conducted usability testing by observing users as they interacted with the game to identify any issues or challenges they faced. Furthermore, we conducted one-on-one discussions with users to gain deeper insights into their experiences and preferences. While playing and testing the game, participants offered valuable insights and feedback on both the positive and negative aspects of the game prototype. They also provided suggestions for improvement. The most notable points are summarized in the following section, where we have also noted our overall impression of the gaming experiment.

Positive Aspects Participants highlighted several positive aspects of the game. They appreciated the innovative mechanics that incorporated musical elements into the gameplay, such as jumping on note blocks and hearing the associated tones, which they found engaging. The interactive audio elements, including the drone tone and the ability to hear a root note for guidance, were well received, especially since players could choose to activate or deactivate this feature. The integration of music and gameplay, like singing to match notes after collecting a fruit, was seen as creative and educational, sparking interest in recognizing and replicating notes. Additionally, the flight mode feature, which allowed players to control their character's vertical position by singing, was positively noted for adding freedom and variety to the gameplay experience.

Negative Aspects Participants encountered several challenges while interacting with the game. Many found the learning curve steep, particularly when trying to understand the mechanics and objectives, which suggests that the instructions were not always clear or intuitive. The controls were also perceived as complex, especially the need to coordinate singing with controller actions such as movement and jumping; this was especially difficult for those without prior experience in jump-and-run games. Additionally, the lack of auditory feedback, such as rewarding sounds for correct actions, made interactions less satisfying. Some participants also commented on the sound design, noting that the drone tones were too aggressive and suggesting that these could be improved by adjusting the wave shape to something softer, between a pure sine tone and a sawtooth.

Suggestions for Improvement Participants suggested several ways to improve the game. First, they recommended providing clearer instructions, particularly regarding the controls and musical elements, to help users better understand how to play. Introducing a guided tutorial that explains how singing interacts with the game mechanics could make the experience more accessible for new players. Participants also noted that more immediate feedback, such as visual aids indicating pitch accuracy or rewards for successfully completing levels, would enhance engagement and motivation. Finally, refining the pitch detection to be more forgiving or accurate was suggested to accommodate players with different levels of singing experience and to give them a greater sense of control.

Overall Impression The game concept is innovative and has the potential to be both engaging and educational. While participants enjoyed the novel mechanics, there were technical and clarity challenges that, if addressed, could greatly enhance the overall player experience. The combination of gameplay and music presents unique opportunities for creativity and learning, making the game appealing to a broad audience.

10 Conclusions and Future Work

In this thesis, we explored how to adapt established offline MIR algorithms into fast, reliable systems suitable for real-time use. We integrated these algorithms into interactive music applications designed to test and demonstrate their practical performance. Although we focused on two specific tasks, beat tracking (`PLP`) and pitch estimation (`SWIPE`), the core methodology we followed is general and can be applied to other MIR tasks as well:

1. **Identify** a well-established and robust MIR algorithm (such as `PLP` or `SWIPE`).
2. **Transform** the algorithm to work in real time, with low latency and efficient computation.
3. **Evaluate** its accuracy under real-time constraints.
4. **Apply** it in interactive music applications to test it under real-world conditions.

Both `PLP` and `SWIPE` were originally designed for offline analysis, but they are built on similar underlying principles. They use Fourier-based representations, kernel correlation, and large analysis windows tailored to the temporal or spectral features they aim to detect. These shared principles made it possible to apply similar transformation strategies, even though the tasks themselves are different. Through this process, we gained a deeper, more nuanced understanding of what enables real-time MIR to function effectively in interactive music applications.

- **Causality** required rethinking how algorithms could preserve meaningful musical information without access to future data, while still responding with zero latency. For example, `PLP` used forward-reaching kernels to provide beat lookahead. This turned out to be a useful feature of its offline design when adapted for real-time use. In contrast, `SWIPE` needed careful alignment of causal windows to reduce delay and maintain responsiveness.
- **Efficiency** involved more than just computational speed. It also required scalability. The algorithms had to process multiple input channels at the same time, especially in ensemble settings. This meant making careful trade-offs between algorithm complexity and the limitations of real-world hardware.
- **Robustness** was important not only for handling background noise and instrument cross-talk, but also for dealing with the instability caused by causal processing itself. For example, `PLP`'s periodicity enhancement helped smooth noisy onset detection. `SWIPE`'s harmonic kernels improved pitch stability in noisy conditions and reduced octave errors.

This transformation from offline analysis to real-time algorithm proved effective for both rhythmic and melodic aspects of music. It enabled real-time performance and uncovered insights and use cases specific to interactive contexts. By adapting established MIR methods in this way, we not only enabled live usability but also revealed new output signals and creative applications that were not apparent in their original offline form. For example, the continuous pulse and stability information from `PLP` proved useful beyond beat detection, serving as control signals for audio effects and enabling new forms of interactive musical feedback. Likewise, the efficient multi-channel implementation of `RT-SWIPE` made it possible to detect subtle tuning inconsistencies during ensemble rehearsals, making it a valuable tool for real-time intonation monitoring and analysis.

Overall, this work demonstrates that established MIR algorithms can be effectively adapted to support novel interactive functionalities in real-time contexts. Limitations such as the use of handcrafted kernels can become creative assets when applied with intention. The development of these systems benefits not only from controlled offline evaluation but also from practical testing in live, interactive environments.

Future Work Beyond beat tracking and pitch estimation, this transformation approach could also be applied to other MIR tasks such as harmony analysis, chord recognition, or structural segmentation, especially when features are based on kernels, time-domain signals, or frequency content. Moreover, the concept of template-based correlation could be integrated with modern machine learning methods. Techniques such as differentiable digital signal processing (DDSP) [42], which combine signal processing with gradient-based learning, allow end-to-end optimization for both perceptual and technical objectives. This opens the door to differentiable templates and hybrid learning systems that could fine-tune kernel structures for specific instruments or performance styles. Early work on `dPLP`[25] and `dSWIPE`[130] points in this direction, combining the efficiency and interpretability of classical algorithms with the trainability of neural methods. Finally, while this thesis focused on interactive music systems, its core ideas such as real-time signal analysis, causal processing, kernel-based matching, and reliable low-latency performance can be applied to many other areas that require real-time processing of audio or time-series data. Possible applications include real-time speech recognition, biomedical signal monitoring, acoustic event detection, and live audio analysis in non-musical settings.

This thesis introduces a practical approach to real-time MIR, informed by real-world application and user feedback. By evaluating concepts through functional interactive systems, it bridges the gap between MIR research and live musical interaction. The work highlights interactivity as a key aspect of musical expression, learning, and creativity, rather than just a technical feature. By addressing challenges such as causality, efficiency, and robustness, this research contributes to the development of real-time MIR systems that are both effective and musically meaningful. In doing so, it brings MIR research closer to the practical needs of musicians, educators, developers, and listeners.

Abbreviations

ADC	analog-to-digital converter
BPM	beats per minute
CD	compact disc
CMMR	Computer Music Multidisciplinary Research
CNN	convolutional neural network
CPU	central processing unit
CQT	constant-Q transform
CREPE	convolutional representation for pitch estimation
CT	continuous-time
DAC	digital-to-analog converter
DAGA	Deutsche Jahrestagung für Akustik
DAW	digital audio workstation
DDSP	differentiable digital signal processing
DFT	discrete Fourier transform
DL	deep learning
DT	discrete-time
EBU	European Broadcasting Union
FAU	Friedrich-Alexander-Universität Erlangen-Nürnberg
FFT	fast Fourier transform
FMP	Fundamentals of Music Processing
FN	false negative
FP	false positive
GI	Gesellschaft für Informatik
GT	ground truth
IBI	inter beat interval
ICMI	Innovative Computer-based Music Interfaces
IIS	Fraunhofer Institute for Integrated Circuits
IP	internet protocol
ISMIR	International Society for Music Information Retrieval
LFO	low frequency oscillator
LSTM	long short-term memory
LUFS	loudness units relative to full scale
MIDI	musical instrument digital interface
MIR	Music Information Retrieval
OS	operating system
OSC	open sound control
PLP	predominant local pulse
RAM	random access memory
RCA	raw chroma accuracy
RNN	recurrent neural network
RPA	raw pitch accuracy
RTF	real-time factor
SMC	Sound and Music Computing Conference
SNR	signal-to-noise ratio
STFT	short-time Fourier transform
SWIPE	sawtooth waveform inspired pitch estimator
TCN	temporal convolutional network
THD	Technische Hochschule Deggendorf

TISMIR	Transactions of the International Society for Music Information Retrieval
TP	true positive
VFA	voicing false alarm
VR	voicing recall

Bibliography

[1] Stefan Balke, Axel Berndt, and Meinard Müller. ChoraleBricks: A modular multitrack dataset for wind music research. *Transaction of the International Society for Music Information Retrieval (TISMIR)*, 8(1):39–54, 2025. doi: 10.5334/tismir.252.

[2] Stefan Balke, Peter Meier, and Meinard Müller. Practicing alone, playing together: A persona-based design approach for amateur wind musicians. In *Proceedings of the Workshop for Innovative Computer-based Music Interfaces (ICMI)*, Chemnitz, Germany, 2025. doi: 10.18420/muc2025-mci-ws06-199.

[3] Juan Pablo Bello and Jeremy Pickens. A robust mid-level representation for harmonic content in music signals. In *Proceedings of the International Society for Music Information Retrieval Conference (ISMIR)*, pages 304–311, London, UK, 2005.

[4] Edgar Berdahl. How to make embedded acoustic instruments. In *Proceedings of the International Conference on New Interfaces for Musical Expression (NIME)*, pages 140–143, London, United Kingdom, 2014. Goldsmiths, University of London. doi: 10.5281/zenodo.1178710. URL `http://www.nime.org/proceedings/2014/nime2014_551.pdf`.

[5] Rachel M. Bittner, Justin Salamon, Mike Tierney, Matthias Mauch, Chris Cannam, and Juan Pablo Bello. MedleyDB: A multitrack dataset for annotation-intensive MIR research. In *Proceedings of the International Society for Music Information Retrieval Conference (ISMIR)*, pages 155–160, Taipei, Taiwan, 2014. doi: 10.5281/zenodo.1417889.

[6] Rachel M. Bittner, Juan José Bosch, David Rubinstein, Gabriel Meseguer-Brocal, and Sebastian Ewert. A lightweight instrument-agnostic model for polyphonic note transcription and multipitch estimation. In *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, Singapore, 2022.

[7] Angel David Blanco, Simone Tassani, and Rafael Ramirez. Real-time sound and motion feedback for violin bow technique learning: A controlled, randomized trial. *Frontiers in Psychology*, 12, 2021. ISSN 1664-1078. doi: 10.3389/fpsyg.2021.648479.

[8] Sebastian Böck and Matthew E. P. Davies. Deconstruct, analyse, reconstruct: How to improve tempo, beat, and downbeat estimation. In *Proceedings of the International Society for Music Information Retrieval Conference (ISMIR)*, pages 574–582, Montreal, Canada, 2020. URL `http://archives.ismir.net/ismir2020/paper/000223.pdf`.

[9] Sebastian Böck and Markus Schedl. Enhanced beat tracking with context-aware neural networks. In *Proceedings of the International Conference on Digital Audio Effects (DAFx)*, pages 135–139, Paris, France, 2011.

[10] Sebastian Böck, Florian Krebs, and Gerhard Widmer. A multi-model approach to beat tracking considering heterogeneous music styles. In *Proceedings of the International Society for Music Information Retrieval Conference (ISMIR)*, pages 603–608, Taipei, Taiwan, 2014.

[11] Sebastian Böck, Filip Korzeniowski, Jan Schlüter, Florian Krebs, and Gerhard Widmer. madmom: A new Python audio and music signal processing library. In *Proceedings of the ACM International Conference on Multimedia (ACM-MM)*, pages 1174–1178, Amsterdam, The Netherlands, 2016. doi: 10.1145/2964284.2973795.

[12] Sebastian Böck, Florian Krebs, and Gerhard Widmer. Joint beat and downbeat tracking with recurrent neural networks. In *Proceedings of the International Society for Music Information Retrieval Conference (ISMIR)*, pages 255–261, New York City, New York, USA, 2016. doi: 10.5281/zenodo.1415835.

[13] Sebastian Böck, Matthew E. P. Davies, and Peter Knees. Multi-task learning of tempo and beat: Learning one to improve the other. In *Proceedings of the International Society for Music Information Retrieval Conference (ISMIR)*, pages 486–493, Delft, The Netherlands, 2019.

[14] Russell Bradford, Richard Dobson, et al. Sliding with a constant q. In *Proceedings of the International Conference on Digital Audio Effects (DAFx)*, pages 363–369, Espoo, Finland, 2008.

[15] Karlheinz Brandenburg, Christian Dittmar, Matthias Gruhne, Jakob Abeßer, Hanna Lukashevich, Peter Dunker, Daniel Gärtner, Kay Wolter, Stefanie Nowak, and Holger Großmann. Music search and recommendation. In Borko Furht, editor, *Handbook of multimedia for digital entertainment and arts*, volume 3, pages 349–383. Springer, New York, USA, 2009. ISBN 0-387-89023-8. doi: 10.1007/978-0-387-89024-1_16.

[16] Andrew Brown. *Computers in Music Education*. Routledge, 2007.

[17] Daniel Brown and Florence Levé. Procedural rhythm transformation for adaptive real-time soundtracks. In *Proceedings of the Sound and Music Computing Conference (SMC)*, pages 326–333, Graz, Austria, 2025. doi: 10.5281/zenodo.15838217. URL `https://doi.org/10.5281/zenodo.15838217`.

[18] Judith C. Brown. Calculation of a constant Q spectral transform. *Journal of the Acoustical Society of America*, 89(1):425–434, 1991. doi: 10.1121/1.400476.

[19] Judith C. Brown and Miller S. Puckette. An efficient algorithm for the calculation of a constant Q transform. *Journal of the Acoustic Society of America (JASA)*, 92:2698–2701, 1992. doi: 10.1121/1.404385.

[20] Christopher J. C. Burges, John C. Platt, and Soumya Jana. Extracting noise-robust features from audio data. In *Proceedings of the International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, pages 1021–1024, Orlando, Florida, USA, May 2002. doi: 10.1109/ICASSP.2002.5743968.

[21] Arturo Camacho and John G. Harris. A sawtooth waveform inspired pitch estimator for speech and music. *The Journal of the Acoustical Society of America*, 124(3):1638–1652, 2008.

[22] Chris Cannam, Christian Landone, and Mark B. Sandler. Sonic Visualiser: An open source application for viewing, analysing, and annotating music audio files. In *Proceedings of the International Conference on Multimedia*, pages 1467–1468, Florence, Italy, 2010.

[23] Chih-Cheng Chang and Li Su. Beast: Online joint beat and downbeat tracking based on streaming transformer. In *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, pages 396–400, Seoul, South Korea, 2024. doi: 10.1109/ICASSP48485.2024.10446611.

[24] Ching-Yu Chiu, Meinard Müller, Matthew E. P. Davies, Alvin Wen-Yu Su, and Yi-Hsuan Yang. An analysis method for metric-level switching in beat tracking. *IEEE Signal Processing Letters*, 29:2153–2157, 2022. doi: 10.1109/LSP.2022.3215106.

[25] Ching-Yu Chiu, Sebastian Strahl, and Meinard Müller. dPLP: A differentiable version of predominant local pulse estimation. In *Proceedings of the International Society for Music Information Retrieval Conference (ISMIR)*, Daejeon, Korea, 2025. doi: 10.5281/zenodo.17706372.

[26] Trevor De Clercq and David Temperley. A corpus analysis of rock harmony. *Popular Music*, 30(1):47–70, 2011.

[27] Alice Clifford and Joshua Reiss. Microphone interference reduction in live sound. In *Proceedings of the International Conference on Digital Audio Effects (DAFx)*, September 2011.

[28] Arshia Cont, Shlomo Dubnov, and David Wessel. Realtime multiple-pitch and multiple-instrument recognition for music signals using sparse non-negativity constraints. In *Proceedings of the International Conference on Digital Audio Effects*, pages 85–92, Bordeaux, France, 2007.

[29] Jiajie Dai and Simon Dixon. Analysis of interactive intonation in unaccompanied SATB ensembles. In *Proceedings of the International Society for Music Information Retrieval Conference (ISMIR)*, pages 599–605, Suzhou, China, 2017.

[30] Matthew E. P. Davies and Sebastian Böck. Temporal convolutional networks for musical audio beat tracking. In *Proceedings of the European Signal Processing Conference (EUSIPCO)*, pages 1–5, A Coruña, Spain, 2019. doi: 10.23919/EUSIPCO.2019.8902578.

[31] Matthew E. P. Davies, Paul M. Brossier, and Mark D. Plumbley. Beat tracking towards automatic musical accompaniment. In *Proceedings of the AES International Conference on Semantic Audio*, Barcelona, Spain, 2005. URL `http://www.aes.org/e-lib/browse.cfm?elib=13124`.

[32] Matthew E. P. Davies, Norberto Degara, and Mark D. Plumbley. Evaluation methods for musical audio beat tracking algorithms. Technical Report C4DM-TR-09-06, Queen Mary University, Centre for Digital Music, 2009.

[33] Matthew E. P. Davies, Adam M. Stark, Fabien Gouyon, and Masataka Goto. Improvasher: A real-time mashup system for live musical input. In *Proceedings of the International Conference on New Interfaces for Musical Expression (NIME)*, pages 541–544, London, United Kingdom, 2014. doi: 10.5281/zenodo.1178744. URL `https://doi.org/10.5281/zenodo.1178744`.

[34] Matthew E. P. Davies, Sebastian Böck, and Magdalena Fuentes. Tempo, beat and downbeat estimation, 2021. URL `https://tempobeatdownbeat.github.io/tutorial/intro.html`.

[35] Alain de Cheveigné and Hideki Kawahara. YIN, a fundamental frequency estimator for speech and music. *Journal of the Acoustical Society of America (JASA)*, 111(4):1917–1930, 2002.

[36] Christian Dittmar, Estefanía Cano, Jakob Abeßer, and Sascha Grollmisch. Music information retrieval meets music education. In Meinard Müller, Masataka Goto, and Markus Schedl, editors, *Multimodal Music Processing*, volume 3 of *Dagstuhl Follow-Ups*, pages 95–120. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, Dagstuhl, Germany, 2012. ISBN 978-3-939897-37-8. doi: http://dx.doi.org/10.4230/DFU.Vol3.11041.95. URL http://drops.dagstuhl.de/opus/volltexte/2012/3468.

[37] Simon Dixon and Emilios Cambouropoulos. Beat tracking with musical knowledge. In *Proceedings of the European Conference on Artificial Intelligence (ECAI)*, pages 626–630, Berlin, Germany, 2000.

[38] Simon Dixon and Gerhard Widmer. MATCH: A music alignment tool chest. In *Proceedings of the International Society for Music Information Retrieval Conference (ISMIR)*, pages 492–497, London, UK, 2005. doi: 10.5281/zenodo.1416952.

[39] Matthias Dorfer, Florian Henkel, and Gerhard Widmer. Learning to listen, read, and follow: Score following as a reinforcement learning game. In *Proceedings of the International Society for Music Information Retrieval Conference (ISMIR)*, pages 784–791, Paris, France, 2018.

[40] Simon Durand, Juan P. Bello, Bertrand David, and Gaël Richard. Robust downbeat tracking using an ensemble of convolutional networks. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, 25(1): 76–89, 2017.

[41] Daniel P.W. Ellis. Beat tracking by dynamic programming. *Journal of New Music Research*, 36(1):51–60, 2007.

[42] Jesse Engel, Lamtharn Hantrakul, Chenjie Gu, and Adam Roberts. DDSP: Differentiable digital signal processing. In *Proceedings of the International Conference on Learning Representations (ICLR)*, Virtual, 2020. URL https://openreview.net/forum?id=B1x1ma4tDr.

[43] Antti J. Eronen and Anssi P. Klapuri. Music tempo estimation with k-NN regression. *IEEE Transactions on Audio, Speech, and Language Processing*, 18(1):50–57, 2010.

[44] Nicholas Evans, Behzad Haki, and Sergi Jordà. Repurposing a rhythm accompaniment system for pipe organ performance. In *Proceedings of the International Conference on New Interfaces for Musical Expression (NIME)*, pages 116–120, Canberra, Australia, 2025. doi: 10.5281/zenodo.15698807.

[45] Sebastian Ewert, Meinard Müller, and Peter Grosche. High resolution audio synchronization using chroma onset features. In *Proceedings of IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, pages 1869–1872, Taipei, Taiwan, 2009. doi: 10.1109/ICASSP.2009.4959972.

[46] Dennis Gabor. Theory of communication. *Journal of the Institution of Electrical Engineers (IEE)*, 93(26): 429–457, 1946.

[47] John M. Geringer, Rebecca B. MacLeod, and Justine K. Sasanfar. In tune or out of tune: Are different instruments and voice heard differently? *Journal of Research in Music Education*, 63(1):89–101, 2015. ISSN 0022-4294. doi: 10.1177/0022429415572025.

[48] Mark Gotham, Rainer Kleinertz, Christof Weiß, Meinard Müller, and Stephanie Klauk. What if the 'when' implies the 'what'?: Human harmonic analysis datasets clarify the relative role of separate steps in automatic

tonal analysis. In *Proceedings of the International Society for Music Information Retrieval Conference (ISMIR)*, pages 229–236, Online, 2021.

[49] Masataka Goto. A real-time music-scene-description system: Predominant-F0 estimation for detecting melody and bass lines in real-world audio signals. *Speech Communication (ISCA Journal)*, 43(4):311–329, 2004.

[50] Masataka Goto and Yoichi Muraoka. A beat tracking system for acoustic signals of music. In *Proceedings of the ACM International Conference on Multimedia*, pages 365–372, San Francisco, CA, USA, 1994.

[51] Masataka Goto, Hiroki Hashiguchi, Takuichi Nishimura, and Ryuichi Oka. RWC music database: Popular, classical and jazz music databases. In *Proceedings of the International Society for Music Information Retrieval Conference (ISMIR)*, pages 287–288, Paris, France, 2002.

[52] Fabien Gouyon, Anssi P. Klapuri, Simon Dixon, Miguel Alonso, George Tzanetakis, Christian Uhle, and Pedro Cano. An experimental comparison of audio tempo induction algorithms. *IEEE Transactions on Audio, Speech, and Language Processing*, 14(5):1832–1844, 2006.

[53] Peter Grosche and Meinard Müller. Extracting predominant local pulse information from music recordings. *IEEE Transactions on Audio, Speech, and Language Processing*, 19(6):1688–1701, 2011. doi: 10.1109/TASL.2010.2096216.

[54] Jiffer Harriman. Start 'em young: Digital music instruments for education. In *Proceedings of the International Conference on New Interfaces for Musical Expression (NIME)*, pages 70–73, Baton Rouge, Louisiana, USA, 2015. doi: 10.5281/zenodo.1179078. URL http://www.nime.org/proceedings/2015/nime2015_218.pdf.

[55] Curtis Hawthorne, Erich Elsen, Jialin Song, Adam Roberts, Ian Simon, Colin Raffel, Jesse H. Engel, Sageev Oore, and Douglas Eck. Onsets and frames: Dual-objective piano transcription. In *Proceedings of the International Society for Music Information Retrieval Conference, (ISMIR)*, pages 50–57, Paris, France, 2018. doi: 10.5281/zenodo.1492341.

[56] Jacob M. Hedges, Robert Sazdov, and Andrew Johnston. Belly of the beast: Insights from developing a VR interactive music experience. In *Proceedings of the Sound and Music Computing Conference (SMC)*, pages 378–385, Graz, Austria, 2025. Institute of Electronic Music and Acoustics, University of Music and Performing Arts Graz. doi: 10.5281/zenodo.15838382. URL https://doi.org/10.5281/zenodo.15838382.

[57] Romain Hennequin, Anis Khlif, Felix Voituret, and Manuel Moussallam. Spleeter: A fast and efficient music source separation tool with pre-trained models. *Journal of Open Source Software (JOSS)*, 5(50):2154, 2020. doi: 10.21105/joss.02154. URL https://doi.org/10.21105/joss.02154.

[58] Mojtaba Heydari and Zhiyao Duan. Beatnet+: Real-time rhythm analysis for diverse music audio. *Transactions of the International Society for Music Information Retrieval (TISMIR)*, 7(1):274–287, 2024. doi: 10.5334/tismir.198.

[59] Mojtaba Heydari, Frank Cwitkowitz, and Zhiyao Duan. BeatNet: CRNN and particle filtering for online joint beat, downbeat and meter tracking. In *Proceedings of the International Society for Music Information Retrieval Conference (ISMIR)*, pages 270–277, Online, 2021. URL https://archives.ismir.net/ismir2021/paper/000033.pdf.

[60] Mojtaba Heydari, Matthew C. McCallum, Andreas F. Ehmann, and Zhiyao Duan. A novel 1D state space for efficient music rhythmic analysis. In *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, pages 421–425, Singapore, Singapore, 2022. doi: 10.1109/ICASSP43922.2022.9747557.

[61] Alex Hofmann, Vasileios Chatziioannou, Alexander Mayer, and Harry Hartmann. Development of fibre polymer sensor Reeds for saxophone and clarinet. In *Proceedings of the International Conference on New Interfaces for Musical Expression (NIME)*, pages 65–68, Brisbane, Australia, 2016. Queensland Conservatorium Griffith University. ISBN 978-1-925455-13-7. doi: 10.5281/zenodo.1176028. URL http://www.nime.org/proceedings/2016/nime2016_paper0014.pdf.

[62] Andre Holzapfel and Thomas Grill. Bayesian meter tracking on learned signal representations. In *Proceedings of the International Society for Music Information Retrieval Conference (ISMIR)*, pages 262–268, New York City, New York, USA, 2016.

[63] Yun-Ning Hung, Ju-Chiang Wang, Xuchen Song, Wei Tsung Lu, and Minz Won. Modeling beats and downbeats with a time-frequency transformer. In *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, pages 401–405, Virtual and Singapore, 2022. doi: 10.1109/ICASSP43922.2022.9747048.

[64] Patrick Edward Hutchings and Jon McCormack. Adaptive music composition for games. *IEEE Transactions on Games*, 12(3):270–280, 2020. doi: 10.1109/TG.2019.2921979.

[65] Cockos Incorporated. REAPER digital audio workstation. https://www.reaper.fm/, 2024.

[66] Jong Wook Kim, Justin Salamon, Peter Li, and Juan Pablo Bello. CREPE: A convolutional representation for pitch estimation. In *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 161–165, Calgary, Canada, 2018. doi: 10.1109/ICASSP.2018.8461329.

[67] Tetsuro Kitahara, Sergio Giraldo, and Rafael Ramírez. JamSketch: A drawing-based real-time evolutionary improvisation support system. In *Proceedings of the International Conference on New Interfaces for Musical Expression (NIME)*, pages 505–506, Copenhagen, Denmark, 2017. doi: 10.5281/zenodo.1176344. URL http://www.nime.org/proceedings/2017/nime2017_paper0101.pdf.

[68] Kris M. Kitani and Hideki Koike. ImprovGenerator: Online grammatical induction for on-the-fly improvisation accompaniment. In *Proceedings of the International Conference on New Interfaces for Musical Expression (NIME)*, pages 469–472, Sydney, Australia, 2010. doi: 10.5281/zenodo.1177827.

[69] Anssi P. Klapuri, Antti J. Eronen, and Jaakko Astola. Analysis of the meter of acoustic musical signals. *IEEE Transactions on Audio, Speech, and Language Processing*, 14(1):342–355, 2006.

[70] Atsuya Kobayashi, Reo Anzai, and Nao Tokui. Exsampling: A system for the real-time ensemble performance of field-recorded environmental sounds. In *Proceedings of the International Conference on New Interfaces for Musical Expression (NIME)*, pages 305–308, Birmingham, UK, 2020. doi: 10.5281/zenodo.4813371. URL https://www.nime.org/proceedings/2020/nime2020_paper58.pdf.

[71] Hendrik Vincent Koops. *Computational Modelling of Variance in Musical Harmony*. PhD thesis, Utrecht University, Utrecht, The Netherlands, 2019.

[72] Filip Korzeniowski and Gerhard Widmer. A fully convolutional deep auditory model for musical chord recognition. In *Proceedings of the IEEE International Workshop on Machine Learning for Signal Processing (MLSP)*, Salerno, Italy, 2016. doi: 10.1109/MLSP.2016.7738895.

[73] Michael Krause, Frank Zalkow, Julia Zalkow, Christof Weiß, and Meinard Müller. Classifying leitmotifs in recordings of operas by Richard Wagner. In *Proceedings of the International Society for Music Information Retrieval Conference (ISMIR)*, pages 473–480, Montréal, Canada, 2020. doi: 10.5281/zenodo.4245472.

[74] Florian Krebs, Sebastian Böck, and Gerhard Widmer. An efficient state-space model for joint tempo and meter tracking. In *Proceedings of the International Society for Music Information Retrieval Conference (ISMIR)*, pages 72–78, Málaga, Spain, 2015.

[75] Florian Krebs, Sebastian Böck, Matthias Dorfer, and Gerhard Widmer. Downbeat tracking using beat synchronous features with recurrent neural networks. In *Proceedings of the International Society for Music Information Retrieval Conference (ISMIR)*, pages 129–135, New York City, New York, USA, 2016.

[76] Marius Kriegerowski and Frank Scherbaum. Pytch: Simultane mehrkanalige Audioanalyse von Gesangstimmen. In *Late-breaking Demos of the Workshop: Musik trifft Informatik at 47. Jahrestagung der Gesellschaft für Informatik*, Chemnitz, Germany, 2017.

[77] Edith L. M. Law, Luis von Ahn, Roger B. Dannenberg, and Mike Crawford. TagATune: A game for music and sound annotation. In *Proceedings of the International Society for Music Information Retrieval Conference (ISMIR)*, pages 361–364, Vienna, Austria, 2007.

[78] Raw Material Software Limited. JUCE framework for audio application and plug-in development. https://juce.com/, 2024.

[79] Robert C. Maher and James W. Beauchamp. Fundamental frequency estimation of musical signals using a two-way mismatch procedure. *The Journal of the Acoustical Society of America*, 95(4):2254–2263, 1994. doi: 10.1121/1.408685.

[80] Ben Maman and Amit H. Bermano. Unaligned supervision for automatic music transcription in the wild. In *Proceedings of the International Conference on Machine Learning (ICML)*, pages 14918–14934, Baltimore, Maryland, USA, 2022.

[81] James A. Mason. Comparison of solo and ensemble performances with reference to pythagorean, just, and equi-tempered intonations. *Journal of Research in Music Education*, 8(1):31–38, 1960. ISSN 0022-4294. doi: 10.2307/3344235.

[82] Matthias Mauch and Simon Dixon. pYIN: A fundamental frequency estimator using probabilistic threshold distributions. In *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 659–663, Florence, Italy, 2014.

[83] Brian McFee, Colin Raffel, Dawen Liang, Daniel P.W. Ellis, Matt McVicar, Eric Battenberg, and Oriol Nieto. Librosa: Audio and music signal analysis in Python. In *Proceedings the Python Science Conference*, pages 18–25, Austin, Texas, USA, 2015. doi: 10.25080/Majora-7b98e3ed-003.

[84] Peter Meier, Gerhard Krump, and Meinard Müller. A real-time beat tracking system based on predominant local pulse information. In *Demos and Late Breaking News of the International Society for Music Information Retrieval Conference (ISMIR)*, Online, 2021. URL https://archives.ismir.net/ismir2021/latebreaking/000021.pdf.

[85] Peter Meier, Simon Schwär, Sebastian Rosenzweig, and Meinard Müller. Real-time MIR algorithms for music-reactive game world generation. In *Proceedings of the Workshop for Innovative Computer-based Music Interfaces (ICMI)*, Darmstadt, Germany, 2022. doi: 10.18420/muc2022-mci-ws03-225.

[86] Peter Meier, Simon Schwär, Gerhard Krump, and Meinard Müller. Evaluating real-time pitch estimation algorithms for creative music game interaction. In *INFORMATIK 2023 – Designing Futures: Zukünfte gestalten*, pages 873–882. Gesellschaft für Informatik e.V., Bonn, Germany, 2023. doi: 10.18420/inf2023_97.

[87] Peter Meier, Simon Schwär, Gerhard Krump, and Meinard Müller. Real-time pitch estimation for creative music game interaction. In *Proceedings of the Deutsche Jahrestagung für Akustik (DAGA)*, pages 1346–1349, Hamburg, Germany, 2023.

[88] Peter Meier, Ching-Yu Chiu, and Meinard Müller. A real-time beat tracking system with zero latency and enhanced controllability. *Transactions of the International Society for Music Information Retrieval (TISMIR)*, 7(1):213–227, 2024. doi: 10.5334/tismir.189.

[89] Peter Meier, Simon Schwär, and Meinard Müller. A real-time approach for estimating pulse tracking parameters for beat-synchronous audio effects. In *Proceedings of the International Conference on Digital Audio Effects (DAFx)*, pages 314–321, Guildford, Surrey, UK, 2024.

[90] Peter Meier, Meinard Müller, and Stefan Balke. A multi-user interface for real-time intonation monitoring in music ensembles. In *Proceedings of the Workshop for Innovative Computer-based Music Interfaces (ICMI)*, Chemnitz, Germany, 2025. doi: 10.18420/muc2025-mci-ws06-202.

[91] Peter Meier, Meinard Müller, and Stefan Balke. Analyzing pitch estimation accuracy in cross-talk scenarios: A study with wind instruments. In *Proceedings of the Sound and Music Computing Conference (SMC)*, pages 3–10, Graz, Austria, 2025. doi: 10.5281/zenodo.15835032.

[92] Peter Meier, Sebastian Strahl, Simon Schwär, Meinard Müller, and Stefan Balke. Pitch estimation in real time: Revisiting SWIPE with causal windowing. In *Proceedings of the International Symposium on Computer Music Multidisciplinary Research (CMMR)*, pages 285–297, London, UK, 2025. doi: 10.5281/zenodo.17496630.

[93] Meinard Müller. *Fundamentals of Music Processing – Using Python and Jupyter Notebooks*. Springer Verlag, 2nd edition, 2021. ISBN 978-3-030-69807-2. doi: 10.1007/978-3-030-69808-9.

[94] Meinard Müller and Frank Zalkow. libfmp: A Python package for fundamentals of music processing. *Journal of Open Source Software (JOSS)*, 6(63):3326:1–5, 2021. doi: 10.21105/joss.03326.

[95] Meinard Müller, Cynthia Liem, Brian McFee, and Simon Schwär. Learning with Music Signals: Technology Meets Education (Dagstuhl Seminar 24302). *Dagstuhl Reports*, 14(7):115–152, 2025. ISSN 2192-5283. doi: 10.4230/DagRep.14.7.115.

[96] Ole Frederik Müermann. Real-time beat tracking for creative music production, 2024. Bachelor Thesis, Friedrich-Alexander-Universität Erlangen-Nürnberg (FAU).

[97] Juhan Nam, Keunwoo Choi, Jongpil Lee, Szu-Yu Chou, and Yi-Hsuan Yang. Deep learning for audio-based music classification and tagging: Teaching computers to distinguish rock from bach. *IEEE Signal Processing Magazine*, 36(1):41–51, 2019. doi: 10.1109/MSP.2018.2874383.

[98] Alan V. Oppenheim, Alan S. Willsky, and Hamid Nawab. *Signals and Systems*. Prentice Hall, 1996.

[99] Yigitcan Özer, Leo Brütting, Simon Schwär, and Meinard Müller. libsoni: A Python toolbox for sonifying music annotations and feature representations. *Journal of Open Source Software (JOSS)*, 9(96):06524:1–6, 2024. doi: 10.21105/joss.06524.

[100] Laurel S. Pardue and Andrew McPherson. Real-time aural and visual feedback for improving violin intonation. *Frontiers in Psychology*, 10, 2019. ISSN 1664-1078. doi: 10.3389/fpsyg.2019.00627.

[101] Danny Perreault, Victor Drouin-Trempe, Vincent Cusson, David Drouin, and Sofian Audry. From performance to installation: How interactive reinforcement learning reframes the roles of performers and audiences. In *Proceedings of the International Conference on New Interfaces for Musical Expression (NIME)*, pages 524–529, Canberra, Australia, 2025. doi: 10.5281/zenodo.15698948. URL `http://nime.org/proceedings/2025/nime2025_76.pdf`.

[102] Silvan D. Peter, Patricia Hu, and Gerhard Widmer. Pairing real-time piano transcription with symbol-level tracking for precise and robust score following. In *Proceedings of the Sound and Music Computing Conference (SMC)*, pages 190–196, Graz, Austria, 2025. Institute of Electronic Music and Acoustics, University of Music and Performing Arts Graz. doi: 10.5281/zenodo.15843588. URL `https://doi.org/10.5281/zenodo.15843588`.

[103] Oriol Romani Picas, Hector Parra Rodriguez, Dara Dabiri, Hiroshi Tokuda, Wataru Hariya, Koji Oishi, and Xavier Serra. A real-time system for measuring sound goodness in instrumental sounds. In *Proceedings of the 138th Audio Engineering Society Convention (AES)*, pages 1106–1111, Warsaw, Poland, 2015.

[104] Graham E. Poliner, Daniel P.W. Ellis, Andreas F. Ehmann, Emilia Gómez, Sebastian Streich, and Beesuan Ong. Melody transcription from music audio: Approaches and evaluation. *IEEE Transactions on Audio, Speech, and Language Processing*, 15(4):1247–1256, 2007.

[105] Thomas Prätzlich, Rachel Bittner, Antoine Liutkus, and Meinard Müller. Kernel additive modeling for interference reduction in multi-channel music recordings. In *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, pages 584–588, Brisbane, Australia, April 2015.

[106] John G. Proakis and Dimitris G. Manolakis. *Digital Signal Processing*. Prentice Hall, 1996.

[107] Colin Raffel, Brian McFee, Eric J. Humphrey, Justin Salamon, Oriol Nieto, Dawen Liang, and Daniel P. W. Ellis. MIR_EVAL: A transparent implementation of common MIR metrics. In *Proceedings of the International Society for Music Information Retrieval Conference (ISMIR)*, pages 367–372, Taipei, Taiwan, 2014.

[108] Alain Riou, Stefan Lattner, Gaëtan Hadjeres, and Geoffroy Peeters. PESTO: Pitch estimation with self-supervised transposition-equivariant objective. In *Proceedings of the International Society for Music Information Retrieval Conference (ISMIR)*, pages 535–544, Milano, Italy, 2023.

[109] Rico Rosenbusch. Automated real-time beat tracking: Response time and confidence analysis, 2024. Bachelor Thesis, Friedrich-Alexander-Universität Erlangen-Nürnberg (FAU).

[110] Sebastian Rosenzweig, Lukas Dietz, Johannes Graulich, and Meinard Müller. TuneIn: A web-based interface for practicing choral parts. In *Demos and Late Breaking News of the International Society for Music Information Retrieval Conference (ISMIR)*, Montreal, Canada, 2020.

[111] Sebastian Rosenzweig, Simon Schwär, and Meinard Müller. libf0: A python library for fundamental frequency estimation. In *Demos and Late Breaking News of the International Society for Music Information Retrieval Conference (ISMIR)*, Bengaluru, India, 2022.

[112] Justin Salamon and Emilia Gómez. Melody extraction from polyphonic music signals using pitch contour characteristics. *IEEE Transactions on Audio, Speech, and Language Processing*, 20(6):1759–1770, 2012. doi: 10.1109/TASL.2012.2188515.

[113] Justin Salamon, Rachel M. Bittner, Jordi Bonada, Juan José Bosch Vicente, Emilia Gómez, and Juan P. Bello. An analysis/synthesis framework for automatic f0 annotation of multitrack datasets. In *Proceedings of International Society for Music Information Retrieval Conference (ISMIR)*, pages 71–78, Suzhou, China, 2017. doi: 10.5281/zenodo.1415588.

[114] Justin Salamon, Rachel Bittner, Jordi Bonada, Juan Jose Bosch, Emilia Gómez, and Juan Pablo Bello. Mdb-melody-synth, 2018. URL `https://doi.org/10.5281/zenodo.1481168`.

[115] Markus Schedl, Hamed Zamani, Ching-Wei Chen, Yashar Deldjoo, and Mehdi Elahi. Current challenges and visions in music recommender systems research. *International Journal of Multimedia Information Retrieval*, 7(2):95–116, 2018.

[116] Frank Scherbaum. On the benefit of larynx-microphone field recordings for the documentation and analysis of polyphonic vocal music. *Proceedings of the International Workshop Folk Music Analysis*, pages 80–87, 2016.

[117] Hendrik Schreiber, Julián Urbano, and Meinard Müller. Global music tempo estimation: Are we done yet? *Transactions of the International Society for Music Information Retrieval (TISMIR)*, 3(1):111–125, 2020. doi: http://doi.org/10.5334/tismir.43.

[118] Hendrik Schreiber, Frank Zalkow, and Meinard Müller. Modeling and estimating local tempo: A case study on Chopin's mazurkas. In *Proceedings of the International Society for Music Information Retrieval Conference (ISMIR)*, pages 773–779, Montréal, Canada, 2020. doi: 10.5281/zenodo.4245546.

[119] Simon Schwär, Sebastian Rosenzweig, and Meinard Müller. A differentiable cost measure for intonation processing in polyphonic music. In *Proceedings of the International Society for Music Information Retrieval Conference (ISMIR)*, pages 626–633, Online, 2021. doi: 10.5281/zenodo.5624601.

[120] Fabian Seipel and Alexander Lerch. Multi-track crosstalk reduction using spectral subtraction. In *Proceedings of the Audio Engineering Society (AES) Convention*, Milan, Italy, 2018.

[121] Victor Shepardson, Jonathan Reus, and Thor Magnusson. Tungnaá: A hyper-realistic voice synthesis instrument for real-time exploration of extended vocal expressions. In *Proceedings of the International Conference on New Interfaces for Musical Expression (NIME)*, pages 536–540, Utrecht, Netherlands, 2024. doi: 10.5281/zenodo.13904943. URL `http://nime.org/proceedings/2024/nime2024_78.pdf`.

[122] Satwinder Singh, Ruili Wang, and Yuanhang Qiu. DeepF0: End-to-end fundamental frequency estimation for music and speech signals. In *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, pages 61–65, Toronto, Canada, 2021. doi: 10.1109/ICASSP39728.2021.9414050.

[123] Julius Orion Smith. *Spectral Audio Signal Processing*. W3K Publishing, http://books.w3k.org/, 2011. ISBN ISBN 978-0-9745607-3-1.

[124] Karolin Stange, Christoph Wick, and Haye Hinrichsen. Playing music in just intonation: A dynamically adaptive tuning scheme. *Computer Music Journal*, 42(3):47–62, 2018. doi: 10.1162/comj_a_00478.

[125] Adam M. Stark. Sound analyser: A plug-in for real-time audio analysis in live performances and installations. In *Proceedings of the International Conference on New Interfaces for Musical Expression (NIME)*, pages 183–186, London, United Kingdom, 2014. doi: 10.5281/zenodo.1178945. URL `https://doi.org/10.5281/zenodo.1178945`.

[126] Adam M. Stark, Mark D. Plumbley, and Matthew E. Davies. Real-time beat-synchronous audio effects. In *Proceedings of the International Conference on New Interfaces for Musical Expression (NIME)*, pages 344–345, New York City, NY, United States, 2007. doi: 10.5281/zenodo.1177249. URL `http://www.nime.org/proceedings/2007/nime2007_344.pdf`.

[127] Domenico Stefani and Luca Turchet. On the challenges of embedded real-time music information retrieval. In *Proceedings of the International Conference on Digital Audio Effects (DAFx)*, pages 177–184, Vienna, Austria, 2022.

[128] Christian J. Steinmetz and Joshua D. Reiss. Efficient neural networks for real-time modeling of analog dynamic range compression, 2022. URL `https://arxiv.org/abs/2102.06200`.

[129] Fabian-Robert Stöter, Stefan Uhlich, Antoine Liutkus, and Yuki Mitsufuji. Open-Unmix – A reference implementation for music source separation. *Journal of Open Source Software*, 4(41), 2019. doi: 10.21105/joss.01667. URL `https://doi.org/10.21105/joss.01667`.

[130] Sebastian Strahl and Meinard Müller. dYIN and dSWIPE: Differentiable variants of classical fundamental frequency estimators. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, 2025. doi: 10.1109/TASLPRO.2025.3581119.

[131] Luca Turchet. Smart musical instruments: Vision, design principles, and future directions. *IEEE Access*, 7: 8944–8963, 2019. doi: 10.1109/ACCESS.2018.2876891.

[132] George Tzanetakis and Perry Cook. Musical genre classification of audio signals. *IEEE Transactions on Speech and Audio Processing*, 10(5):293–302, 2002. doi: 10.1109/TSA.2002.800560.

[133] Jian-Heng Wang, Siang-An Wang, Wen-Chieh Chen, Ken-Ning Chang, and Herng-Yow Chen. Real-time pitch training system for violin learners. In *Proceedings of the IEEE International Conference on Multimedia and Expo Workshops (ICMEW)*, pages 163–168, New York, USA, 2012. doi: 10.1109/ICMEW.2012.35.

[134] Christof Weiß, Sebastian J. Schlecht, Sebastian Rosenzweig, and Meinard Müller. Towards measuring intonation quality of choir recordings: A case study on Bruckner's Locus Iste. In *Proceedings of the*

International Society for Music Information Retrieval Conference (ISMIR), pages 276–283, Delft, The Netherlands, 2019.

[135] Matthew Wright, Adrian Freed, and Ali Momeni. OpenSound control: State of the art 2003. In *Proceedings of the International Conference on New Interfaces for Musical Expression (NIME)*, pages 153–159, Montreal, Canada, 2003.

[136] Yu-Te Wu, Berlin Chen, and Li Su. Multi-instrument automatic music transcription with self-attention-based instance segmentation. *IEEE/ACM Transactions on Audio, Speech and Language Processing*, 28:2796–2809, 2020. doi: 10.1109/TASLP.2020.3030482.

[137] Guangyu Xia and Roger Dannenberg. Improvised duet interaction: Learning improvisation techniques for automatic accompaniment. In *Proceedings of the International Conference on New Interfaces for Musical Expression (NIME)*, pages 110–114, Copenhagen, Denmark, 2017. doi: 10.5281/zenodo.1176189.

[138] Shu-Nung Yao. Audio effect units in mobile devices for electric musical instruments. *IEEE Access*, 7: 159239–159250, 2019. doi: 10.1109/ACCESS.2019.2950780. URL https://doi.org/10.1109/ACCESS.2019.2950780.

[139] Jingwei Zhao, Gus Xia, and Ye Wang. Beat transformer: Demixed beat and downbeat tracking with dilated self-attention. In *Proceedings of the International Society for Music Information Retrieval Conference (ISMIR)*, pages 169–177, Bengaluru, India, 2022. URL https://archives.ismir.net/ismir2022/paper/000019.pdf.

[140] Udo Zölzer. *Digital Audio Signal Processing*. John Wiley & Sons, Hoboken, New Jersey, 2nd edition, 2008.

[141] Udo Zölzer. *DAFX: Digital Audio Effects*. John Wiley & Sons, 2nd edition, 2011. ISBN 0470665998.