PIANO SCORE FOLLOWING WITH HIDDEN TIMBRE OR TEMPO USING SWITCHING KALMAN FILTERS

Yucong Jiang

Submitted to the faculty of the University Graduate School

in partial fulfillment of the requirements

for the degree

Doctor of Philosophy

in the Luddy School of Informatics, Computing, and Engineering

Indiana University

August, 2020

Accepted by the Graduate Faculty, Indiana University, in partial fulfillment of the requirements for the degree of Doctor of Philosophy.

Doctoral Committee:

Christopher Raphael, Ph.D.

David Crandall, Ph.D.

Minje Kim, Ph.D.

Daniel McDonald, Ph.D.

Date of Defense: July 31, 2020

Acknowledgements

First and foremost, I would like to acknowledge my dissertation advisor, Chris Raphael, who provided essential insights and resources for this project. I thank him for all the help and guidance he has provided during my Ph.D. program; I've learned to think critically, patiently, and independently.

I would also like to express my sincere appreciation to the other members of my doctoral committee, David Crandall, Minje Kim, and Daniel McDonald, who offered valuable advice not only on research, but also professional development.

I thank Gerhard Widmer for pointing me to the then-newly published MAESTRO data set, which has been a tremendous help on this project. I thank Daniel Manrique-Vallier for his *Bayesian theory and data analysis* course; I thoroughly enjoyed that course, and it helped me develop important intuitions toward my research. I thank Joan Middendorf for her pedagogy course—the theories and skills I learned from that course powerfully influenced the way I communicate ideas to my readers and students. I would also like to take this opportunity to express my gratitude to my math teachers from years past, Qiulian Yuan, Jianbing Tang, and Xiuqing Chen; their teaching truly stimulated my interest in math, and prepared me to meet the challenges of graduate school.

I am grateful to the hosts of my two Google internships, Taghrid Samak and Jen Iofinova. The working experience at Google not only made me a better programmer, but also a better communicator and a more well-rounded professional. I am also genuinely grateful to my previous mentors from WiMIR, Doug Turnbull, Mohamed Sordo, and Mark Cartwright, whose support and advice had important impacts on both my studies and my career decisions.

I'm glad to have shared an office with Liang Chen, Yupeng Gu, Jin Rong, and Sanna Wager: I not only enjoyed our research-related discussions, but also our friendship over the

years. I appreciated the company and support from many other good friends including Chi Chen, Wen Chen, Mingfei Gao, Eman Hassan, Can Liu, Xin Li, Jie Ma, and Yu Xu. I thank the superb teaching by my fantastic piano teacher, Patrick Elisha, which helped me maintain a sense of composure to balance the pressures of graduate work. For similar reasons, I value Bloomington's active and friendly social dancing community, including SoInKiz, IU Swing Dance Club, and Ritmos Latinos Indiana. The good times I spent either in the piano practice rooms or on the dance floor truly enriched my life during my Ph.D. years.

I could not have done this without the unconditional support of my parents for all my intellectual endeavors and adventures, including writing this dissertation.

Last but not least, I'm greatly indebted to my partner, Chris Miller, for his indispensable support, patience, and understanding over the past few years. I thank him for many great times we've shared, and for his company and support during some challenging times, including the ongoing COVID-19 pandemic that added such drama to the final act of my graduate school experience.

Yucong Jiang

PIANO SCORE FOLLOWING WITH HIDDEN TIMBRE OR TEMPO
USING SWITCHING KALMAN FILTERS

Score following is an AI technique that enables computer programs to "listen to" music: to track a live musical performance in relation to its written score, even through variations in tempo and amplitude. This ability can be transformative for musical practice, performance, education, and composition. Although score following has been successful on monophonic music (one note at a time), it has difficulty with polyphonic music. One of the greatest challenges is piano music, which is highly polyphonic. This dissertation investigates ways to overcome the challenges of polyphonic music, and casts light on the nature of the problem through empirical experiments.

I propose two new approaches inspired by two important aspects of music that humans perceive during a performance: the pitch profile of the sound, and the timing. In the first approach, I account for changing timbre within a chord by tracking harmonic amplitudes to improve matching between the score and the sound. In the second approach, I model tempo in music, allowing it to deviate from the default tempo value within reasonable statistical constraints. For both methods, I develop switching Kalman filter models that are interesting in their own right. I have conducted experiments on 50 excerpts of real piano performances, and analyzed the results both case-by-case and statistically.

The results indicate that modeling tempo is essential for piano score following, and the second method significantly outperformed the state-of-the-art baseline. The first method, although it did not show improvement over the baseline, still represents a promising new direction for future research. Taken together, the results contribute to a more nuanced and multifaceted understanding of the score-following problem.

Christopher Raphael, Ph.D.

David Crandall, Ph.D.

Minje Kim, Ph.D.

Daniel McDonald, Ph.D.

Table of Contents

List of Tables

List of Figures

CHAPTER 1

Introduction

> "The primary goal of Artificial Intelligence is to make machines smarter. The secondary goals of Artificial Intelligence are to understand what intelligence is [,] . . . and to make machines more useful . . . "
>
> — Winston and Brown (1979)

1.1 Music & Machine Intelligence

Though it might take a long time for humans to understand *what intelligence is*, we have already built machines that appear to be intelligent: machines that can identify objects (e.g., birds) in an image, machines that can translate from one language to another, and machines that can diagnose diseases. These machines imitate human sensory ability and behavior in terms of vision, language, and inference. Another amazing ability humans have is to listen to and make music, and this, too, has inspired a category of research that demonstrates such musical intelligence. This is the challenge that motivates this dissertation: building a machine that can keep track of a musical performance in relation to its written score, even through variations in tempo and amplitudes—a score follower.

It is natural for humans to appreciate music. Most of us can follow the beat and rhythm in music, anticipate when the next note will happen, identify the genre and instruments (maybe after a little training), and feel the mood in music. With some musical training, humans can perform or compose music. Researchers have tried to build machines that show these musical capacities: examples include beat tracking (Peeters and Papadopoulos, 2011), automatic music transcription (Benetos et al., 2013), computer accompaniment systems (Dannenberg and Raphael, 2006), music recommendation systems (Pandora, Spotify, and Youtube Music), and automatic music composition (Fernández and Vico, 2013). This

kind of research is interdisciplinary, and usually involves techniques from computer science, audio signal processing, statistics, music theory, and sometimes psychology. Score following—the focus of this dissertation—stands out as a related problem that can inspire a variety of applications.

1.2 The Score-following Problem

Imagine someone is reading a book to you, and you have the script (the book) at hand. It is an easy game for you to trace the word being spoken at each moment. Now replace the book with a musical score, and replace the reading with a performance of the score. Similarly, with some musical training, humans can identify which note is sounding in the score at each moment. The score-following problem involves building a computer program that can trace the musical events in a given musical score during a live performance of that piece. This task is also called the "online audio-to-score alignment", which tracks the current position in the score while the performance is going on. For the scope of this dissertation, I focus on single-instrument music in the common-practice period, especially piano music, although similar methodologies and their implications can be generalized to some other forms of music as well.

To use the "language" of computers, the performance is represented by an audio file, and the musical score is represented in symbolic formats which maintain the abstraction of the score, e.g., a MIDI[1] file. The goal is to figure out which part of the audio (in terms of seconds) corresponds to which musical position (e.g., beats) in the score. In other words, we want to synchronize (align) the sound (the audio) and the script (the score). Figure 1.1 illustrates this audio-to-score alignment problem.

Musical scores only specify the *nominal length* (musical length) of each note; e.g., a quarter note occupies one beat in $\frac{4}{4}$ meter music. The actual *performed length* of a note

[1]MIDI (Musical Instrument Digital Interface) does not store the sound itself, but stores the "instructions" to recreate the sound by digital devices: specifying exactly when and how electronic circuits should turn on and off (Rothstein, 1995). It is a level of abstraction between music sound and printed sheet music.

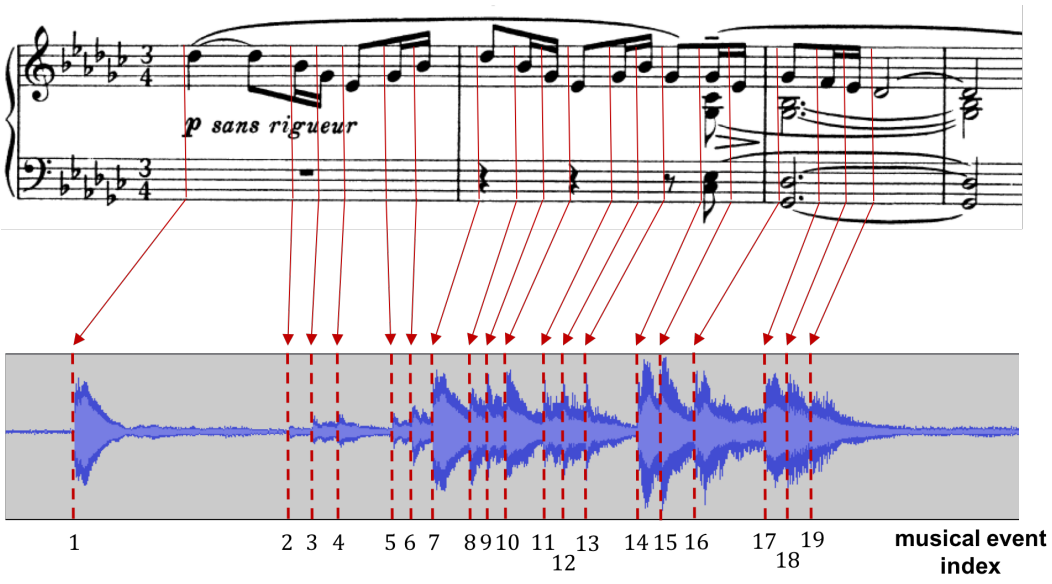


Figure 1.1: Illustration of the audio-to-score problem (Jiang and Raphael, 2019). The upper panel is the music score; the lower panel is the audio (performance of this score). The dashed lines in the audio are the beginning moments of the musical events in the score. There are 19 events in this excerpt.

is uncertain for two reasons. First, the general tempo of a piece is unknown. The score only indicates a range of recommended tempi; e.g., *Andante* (at a walking pace) usually is between 76 and 108 bpm (beats per minute). Therefore, we cannot know how much time a quarter note would actually last in a performance. Second, the tempo can change throughout the piece. Examples are tempo direction *ritardando* (a gradual decrease of tempo) and *rubato* (varying the tempo freely, usually without changing the overall pace). In addition, performers sometimes intentionally deviate from the general tempo, expressing their personal interpretations of some notes. Actually, most performances that stick to a fixed tempo are perceived to be boring and far from the intention of music. Both factors—unknown general tempo and deviations from the general tempo—make it difficult to anticipate how long each note would actually last.

1.2.1 Offline Alignment

A related problem to score following is called "*off*line audio-to-score alignment", in contrast to the other name for score following, which is "online audio-to-score alignment". Both problems aim to align the audio with the score, but they differ in when the alignment happens. Offline alignment starts the task *after* the performance is done, allowing the program to access the entire audio recording. Online alignment constantly figures out the current position in the score while the performance is going on, so the program can only access the audio data received before the current moment.

Since the two scenarios access different amount of data, they are also suited to different algorithmic styles. Offline alignment often uses offline algorithms which are allowed to look at the entire data. Online alignment, on the other hand, often adopts online algorithms which process the input data piece by piece. Online algorithms form a moment-by-moment understanding of the score position based on the data received before the current moment (though this understanding could change as more data arrive), and these understandings might not be optimal considering the entire data (as available offline). Therefore, offline alignment is generally more accurate than online alignment, and it is considered as an easier task. Despite their difference in algorithm styles, the two problems can share similar methodologies, as can be seen in Section 1.3.1.

1.2.2 Useful Applications

Online Alignment Applications

Score following is the foundation of many useful applications. It forms the heart of any musical score page turner (Arzt et al., 2008). Score following also serves as a crucial layer of automatic accompaniment systems (Dannenberg and Raphael, 2006): a soloist can be followed and accompanied by a computer which might play the prerecorded orchestra part while adapting to the speed of the soloist. These kinds of systems enable a violin soloist,

for example, to rehearse a violin concerto with a computer, relieving the pain of assembling dozens of orchestra players to rehearse with him or her.

Some composers use score-following systems as a composition tool, creating virtual scores, scores that consist of electronic programs which react to a live performance (Cont, 2011; Ensemble and Suurmond, 2001). The outcome of the virtual score is a function of the live performance—the computer "listens to" the ongoing performance (according to a score) and decides when and how to trigger the programs that would generate certain sound.

Score following can also enable real-time audio enhancement applications: processing the input audio and outputting the enhanced audio in real-time; e.g., auto-tune in a live performance. In addition, score following can provide live feedback about a performer's playing at the signal level, which can be further developed into a music-education tool, such as a computer tutor (Schoonderwaldt et al., 2005).

All these applications share the component of bridging the musical sound with the musical intentions in the score, and score following will open up such new opportunities for future applications that we have not thought of yet.

Offline Alignment Applications

Since offline alignment doesn't require real-time music performance, it can process large amounts of audio data in the background. With audio aligned with notes in the scores, we can query specific audio segments by specifying the notes in the scores, and we can locate and play back recordings from any requested notes in the score, making music editing or post processing more convenient. For music-education purposes, we can give feedback on a performance at the signal level by analyzing the audio of known notes: how long each note is played; how the tuning, pitch, timbre, and vibrato vary. As a research tool for *computational musicology*, aligned recordings would also aid empirical studies on musical performance (for example, Repp (1990), which used manually aligned recordings to study the patterns of expressive timing by nineteen famous pianists). Imagine if every segment

of performance recordings in a music database were annotated with its score position—it would open up opportunities and lead to valuable potential for large-scale analysis and indexing on music audio.

1.3 Historical Background & Limitations

1.3.1 History of Research

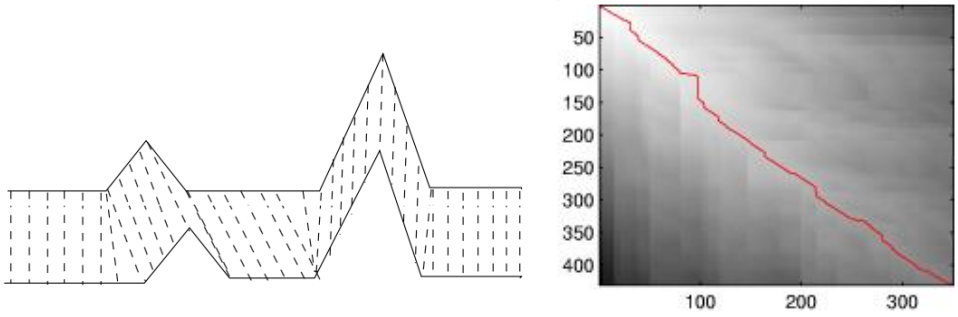
In 1984, score-following research started to gain attention because of two independent papers: Dannenberg (1984) and Vercoe (1984). At that time, computers were not fast enough to process real-time audio generated from the instruments. Both authors installed sensors on instruments to generate *symbolic* input (predefined events) for the computers. Such systems are not flexible enough to deal with errors or variant expressions by different performers. As computers got faster in the early 1990s, researchers started to use raw audio as the input, allowing musicians to use their own instruments (Cont, 2010). Many systems (e.g., Puckette and Lippe (1992)) then depended on the result of an intermediate step, called *pitch detection*, which estimates the musical pitch information of a segment of sound and classifies it within a finite set of predetermined pitch categories; the score follower, then, relies on the pitch-detection result to further infer the score positions. However, pitch-detection algorithms are prone to error (Orio et al., 2003), and these errors could propagate from the pitch-detection step to the score-following results. Since the late 1990s, stochastic approaches (as represented by Grubb and Dannenberg (1997) and Grubb (1998)) have dominated this topic because they are more robust than all previous methods, and they no longer depend on pitch detection but can directly deal with audio signals. Many of these approaches are based on the hidden Markov model (HMM).

The HMM is one of the most well-known *state-space models*: models that use state variables to represent the temporal evolution of a dynamical system. The HMM operates on discrete states, and on pre-defined *transition probabilities* among these states. Transition probabilities encode our knowledge about how the state sequence might progress, from

beginning to end, *before* observing any data—therefore, this part of the HMM design is referred to as the *prior model*. In the audio-to-score alignment problem, our only prior knowledge about the musical performance (the data) is the musical score, and the states at different times represent the score position information that we want to infer. The other part of the HMM is called the *data model*. The data model evaluates how different hypotheses are supported by the observed data at that time; it uses a *data likelihood* function so that hypotheses more consistent with the observation get higher results than those contradictory to the observation. It is also common to extract representative features from the data, and use those features (instead of raw data) as the observation in the data likelihood function. Combining the results from the prior model and the data model, we can either infer the most likely state sequence (score positions) in the offline scenario—the "most likely path" problem—or, in the online scenario, infer the distribution over multiple hypotheses at a moment given the observations available until this moment—the "filtering" problem. Both problems can utilize dynamical programming techniques to reduce the computation complexity. In 1999, Raphael presented the use of HMMs to address both offline and online alignment problems (Raphael, 1999). That framework is highly relevant to some later works by other researchers such as Orio and Déchelle (2000) and Cont (2006). Based on am HMM, Raphael's *Music Plus One* (Raphael, 2010) is one of the state-of-the-art musical accompaniment systems, and it has been tested and used in live concerts. This system has been refined over the past two decades, and it works well on monophonic music; it is also the baseline of my work here.

Starting from 2001, many researchers have used the dynamic time warping (DTW) method, especially on offline alignment, including Orio and Schwarz (2001), Soulez et al. (2003), and Müller et al. (2004, 2006). DTW is an algorithm used for matching two sets of time-series data that may progress with different speeds, as illustrated in Figure 1.2a. A match is usually monotonic, meaning the lines in the figure must not overlap. A matching result can also be visualized as a connected path in a matrix starting from the top left

to the bottom right, as in Figure 1.2b. Every path is associate with a total cost coming from two sources: 1) a horizontal or a vertical advance increases the cost by one, and a diagonal advance increases the cost by two; 2) the cost increases with the distance of every pair of elements—usually an Euclidean distance of two features vectors in the same space. We can calculate the optimal path that has the lowest cost using dynamical programming techniques. In the offline alignment problem, both the score and the audio are converted to the same feature space, so the matching distance can be calculated. DTW is entirely feature-based, without any model parameters as in the HMM. DTW was later applied to online alignment too, but it showed poorer performance than HMM-based models according to the experiments by Cont (2010), which compared the HMM-based approach with DTW-based approaches from Dannenberg and Hu (2003) and Dixon (2005). One reason why DTW might not be preferable for online alignment is that the dynamical programming in DTW is based on the "best path" (one most likely path), while the HMM is based on "filtering" (summary of all previous paths). What score following emphasizes is really "where the performance is at the current moment" rather than "what is the most likely path leading to the current moment".



(a) Matching two sequences. (b) Matching path in a matrix.

Figure 1.2: Illustration of DTW. ((a) from en.wikipedia.org/wiki/Dynamic_time_warping; (b) from ee.columbia.edu/~dpwe/resources/matlab/dtw)

In the 2010s, more researchers have chosen to model *tempo* in their methods, for either offline or online alignment. Raphael (2004) models the tempo as a continuous state variable in a state-space model, and thus it involves both continuous and discrete states—which

Raphael called a "hybrid" graphical model. This algorithm was tested on aligning orchestra music offline, and it represents one of the earliest works that successfully modeled tempo on this topic. Another state-of-the-art system (besides Music Plus One), *Antescofo*, uses hidden *semi*-Markov models which are described in Cont (2008, 2010) and Cuvillier (2016). That approach does not directly model tempo, but models the distribution of *note lengths* instead. Antescofo also serves as a synchronous programming language for composing virtual scores (described in Section 1.2.2) used by a variety of composers (Cont, 2011). Among other works that model tempo are Joder et al. (2010), who first adopt conditional random field; Montecchio and Cont (2011), Otsuka et al. (2011), and Duan and Pardo (2011), who use continuous state-space models, and use particle filters to approximate the inference results; and Arzt and Widmer (2010), who incorporate learned tempo information from multiple performances into an online-DTW algorithm.

The data features used in different systems can vary, but they mostly fall into the pitch or frequency domain. For example, the *chroma* feature contains 12 pitch categories that correspond with the equal-tempered chromatic scale (the 12 piano keys in any octave). However, all information represented by these various features is fully contained in the Fourier transformation result, and I use the modulus of the result as the feature. İzmirli and Dannenberg (2010) and Agrawal and Dixon (2019) provide additional insights on music-related feature engineering, venturing beyond the scope of this research.

Recently, a variation of the audio-to-score alignment problem appeared: aligning audio from a music practice session instead of from a performance. In a typical music practice session, the musician might repeat certain difficult measures or notes to work on them. The musician might also pause after a note, and pick a previous or later note and start playing from there. Since the musician is allowed to skip notes and jump between places in the score, we can no longer assume the audio aligns with the score in a linear fashion like in the regular alignment problems. This variation massively increases the search space of the problem, thus requiring a clever way to balance computation complexity and accuracy. This

variant problem could have offline and online versions too. Nakamura et al. (2013, 2016) use a HMM-based model to address the online version of this problem on monophonic music (one note at a time). Jiang et al. (2019) also choose an HMM-based model but for the offline scenario, and have tested it on clarinet, piano, and viola music. Pardo and Birmingham (2005) deal with a similar scenario, but the playing can only repeat or skip at specified sections in the score. Table 1.1 compares various versions of the audio-to-score alignment problem, and shows the focus of this dissertation in a broader context.

Performance Scenarios	**Problem Scenarios**
linearly performed music	offline alignment ***online alignment (score following)***
practice music	offline alignment online alignment (score following)

Table 1.1: Variations and aspects of audio-to-score alignment problems. The categories that correspond with the focus of this text are bolded and italicized.

1.3.2 Limitations of Existing Research

Over the years, score following has been successful on *monophonic music*—music played with one note at a time (e.g., music played on a clarinet or a violin). However, following *polyphonic music* (music allowing simultaneous notes) is still an open problem. One of the most challenging cases is piano music, which can have four or more notes sounding at the same time. The two state-of-the-art systems, Music Plus One and Antescofo, have been successfully used to follow soloists in live concerts (mostly on monophonic instruments), but neither of them is robust on piano music. Few works have been focused on following piano music, despite the fact that piano is one of the most popular instruments.

Score following is a harder problem than offline alignment because it requires inference from *partially* available data. Score following is also more complicated to evaluate. In offline alignment, the entire data are available; the goal is to find the most plausible note *onset* times in the audio (the moment when a note starts) based on the entire data, and so

every segment of audio would have one corresponding event in the score. However, in score following, the result need not be note onsets, because the program might change its belief about an onset after it receives more data. In fact, if the score-following program tries to detect onsets, the longer the program waits before detecting an onset, the more data become available, and the more accurate the detection is: this causes a trade-off problem between accuracy and promptness. Unlike in offline alignment where the errors can be inferred directly from the detected onsets, there is no standard evaluation method on the score following problem, which makes it difficult to track improvements or compare different systems. Cont et al. (2007) first proposed an initial framework for assessing the robustness and preciseness of a score-following system from a number of measurements such as error rates, average latency, and piece completion. However, that approach involves ambiguity due to the trade-off between accuracy and promptness: allowing more delays would generally lead to better accuracy, because more data are observed during the delay.

Testing data is not standardized either. Some researchers choose to test on synthesized audio from MIDI instead of recordings from musicians, mostly to avoid the trouble of getting the ground truth. However, from my experiments, I find that synthesized audio is much easier to align than real performance recordings of the same piece (which are the ultimate goal). Real performance is more complicated and unpredictable with a variety of situations which are not easily simulated. In addition to that problem, quite a few studies test only on a small number of pieces, test only on monophonic pieces, present marginal results, or do not include a statistical report of results at all.

Following piano music is still an open problem for machines, even though it is an easy task for humans. Without a better understanding about why it is especially challenging, we will be stuck at the current bottleneck, and will sacrifice a series of valuable applications that depend on the score-following component, as discussed in Section 1.2.2.

1.4 Motivations

1.4.1 Why is Following Polyphonic Music More Difficult?

Score following on polyphonic music is generally more difficult than on monophonic music; most methods successful with monophonic music can suffer on polyphonic music. Researchers usually rely on the *spectral feature* of the sound to distinguish different notes. The spectral feature of any segment of audio can be calculated from the Fourier transformation, and the result is a *frequency spectrum* revealing what frequencies exist in the sound and how much of them. Such spectral features are especially useful on musical sound because musical notes have direct relationship with frequencies. Every note contains a *fundamental frequency*, also referred to as "first harmonic", defining its distinguished sound (fourth column in Figure 1.3), e.g., 440Hz for A4. In addition, a note also has frequencies at multiples of the fundamental frequency, forming spectral peaks around those frequencies; these are "higher harmonics" of this note. For example, in the *standard A* (A4) note, we expect to observe peaks around frequencies 440Hz, 880Hz, 1320Hz, . . . as in Figure 1.4a. The relative intensities among these harmonics affect the timbre aspect of the sound.

Every note has its designated locations for peaks in a spectrum, which can help identify what note is sounding. Although the spectrum profile of the same note can vary among different instruments as they have different timbres (so different relative intensities among peaks), a note's spectrum does not change much within an instrument. Therefore, we can design a fixed spectrum template for each note within an instrument, without deviating too much from reality. However, if we allow more than one note to sound at the same time, as in polyphonic music, the spectrum profile of a *chord* is not so predictable any more. The profile can vary considerably according to the relative loudness among the notes in the chord, which is decided by the performer. Figure 1.4 shows an example of a two-note chord, A4 and D4, played with different loudness ratios, and resulting in different spectrum profiles. Therefore, in polyphonic music, the observed data (spectral features) are more

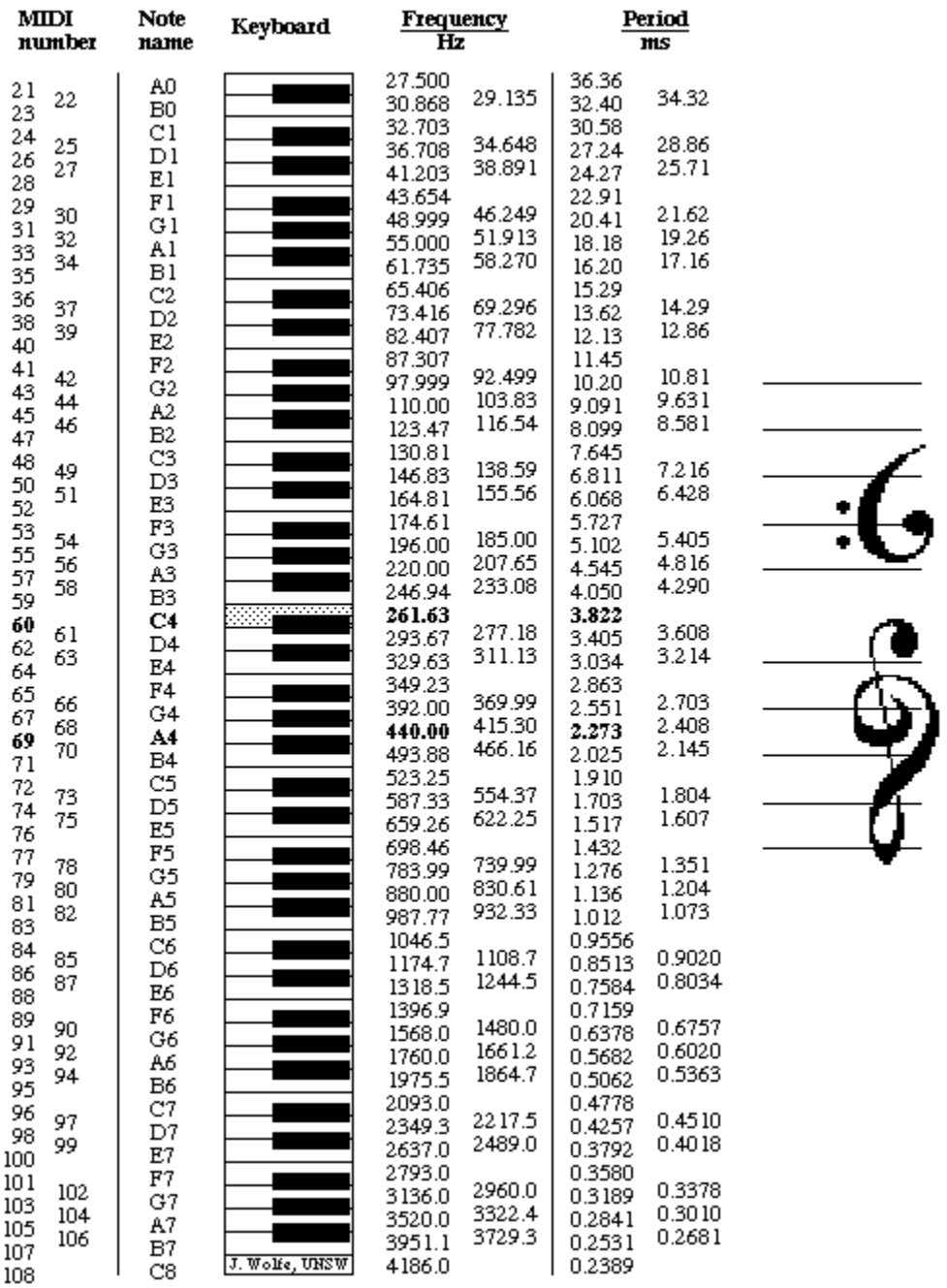
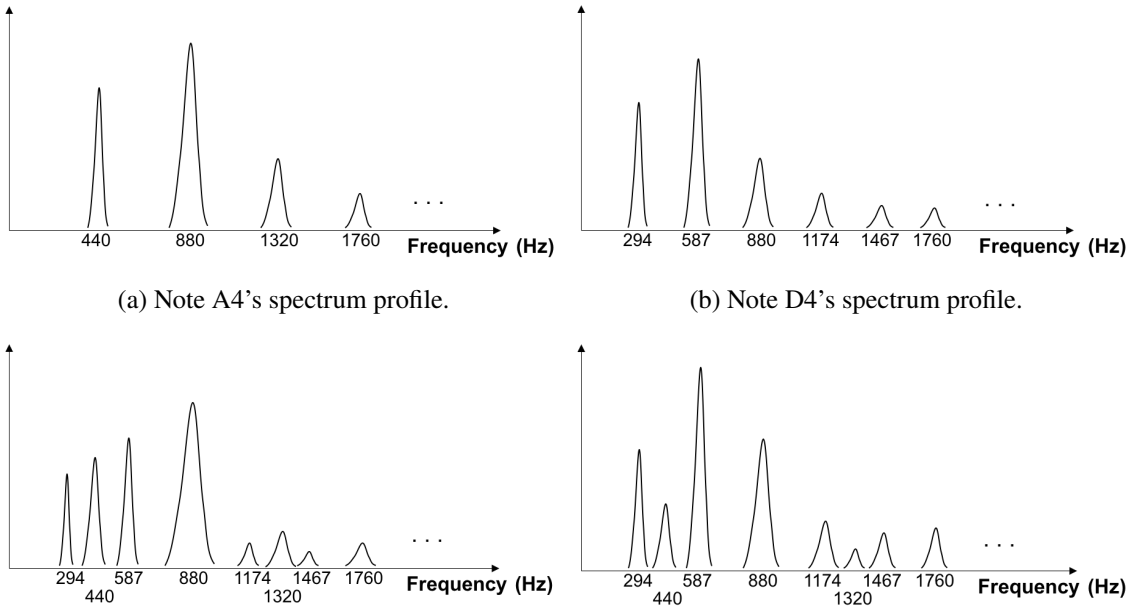


Figure 1.3: Piano notes and fundamental frequencies. (from newt.phys.unsw.edu.au/jw/notes.html)



(a) Note A4's spectrum profile.

(b) Note D4's spectrum profile.

(c) The spectrum profile of A4 mixed with D4 by the ratio of 1:1.

(d) The spectrum profile of A4 mixed with D4 by the ratio of 1:2.

Figure 1.4: (a) and (b) are the frequency profiles of two different individual notes, A4 and D4, respectively. (c) and (d) are the frequency profiles of the chord including notes A4 and D4, played with two different relative loudness ratios.

difficult to predict given the notes in the score—in other words, it is more difficult to model the data accurately than in monophonic music.

Most of the time, a score-following program's efforts are spent on distinguishing among neighboring chords in the score, to infer when the performer has (or has not) moved to the next chord. It is common in piano music that neighboring chords share some notes (held notes), or at least share some harmonics (e.g., notes A4 and D4 share the harmonics that are multiples of 880Hz). This makes the neighboring chords' profiles less distinguishable from each other because they both have peaks around certain frequencies (though probably with different energy). Therefore, methods that work well on monophonic music could fail to distinguish those "similar" chords in piano music.

In addition, as the piano is a percussion instrument, its sound decays over time, with different rates among different frequencies—higher frequencies usually decay faster than lower frequencies. This causes a chord's frequency spectrum to evolve over time, making

it even more difficult to decide the chord's profile template.

Last but not least, a score-following program tends to struggle at places where a sustain pedal is applied. *Sustain pedaling*, oftentimes simplified as "pedaling", is very common in piano music: when the pedal is pressed, the dampers of played keys are kept away from the strings so that these notes can continue to sound even after their keys are released. It brings trouble when the pedal extends sound from the previous note to the next note, or even to later notes, because this can cause a mismatch between the audio and the musical score—what the program depends on to anticipate sound profiles. In general, musical scores grant a high degree of flexibility about when notes end—it is common to use pedaling to delay note *off*sets. The program must be robust despite all kinds of audio-score mismatches: notes ending late because of pedaling, notes ending early or late because of personal interpretations, and occasional mistakes (wrong notes, extra notes, or missed notes).

1.4.2 New Methods

Most score-following methods share the same general idea: possible performances are viewed as paths through a state graph which is derived from the musical score. For any moment in the audio, a score-following algorithm infers the current state of the performance—score positions—from the available audio data up to that moment. The algorithm does this by evaluating the hypothesis paths through the state graph, each path representing a possible performance. From the Bayesian point of view, the evaluation criteria for a path include two aspects: 1) how much a hypothesis is consistent with the *prior model*, which represents the anticipation of the musical performance *before* observing any data, and 2) how much the observed data support a hypothesis, which is the *data model*. Since the prior model essentially is about *timing*—when a note appears and how long it lasts, it is also referred to as the *timing* model. In this dissertation, I propose two ideas to improve these two aspects respectively.

Improved Data Model

Existing algorithms share the assumption that a note's spectrum profile is *fixed* throughout its lifetime, but that is not true. The piano sound decays over time, with different decay rates among frequencies, resulting in a changing frequency spectrum with time. If we can update the spectrum templates according to the newly observed data, we can achieve a more accurate data model. For a chord in the score, we know *where* (in terms of frequencies) to expect harmonic peaks in the sound, but cannot be sure about their amplitudes. The Kalman filter (Kalman, 1960) would be suitable for estimating such continuous variables that change gradually with time, by taking the observed data into account. Tracking the amplitudes of the harmonic peaks will improve the spectrum templates, and thus potentially improve the overall results. Tracking the harmonics will also enable *data-driven* spectrum templates where the chord templates can adapt to the current data (sound), thus avoiding the problem of deciding relative note-loudness ratios within a chord (discussed in Section 1.4.1). For example, if a harmonic peak is underestimated at the beginning of a note, the Kalman filter will adjust its estimation and expect larger amplitudes later on to match the observed amplitude value. However, we do not know, in advance, the boundary of every chord in the audio; rather, these score positions are exactly what we are trying to infer. Therefore, there should be multiple Kalman filters, each tracking a different possible scenario—this is called a *switching Kalman filter* model. The number of Kalman filters needed increases exponentially with time, because every state path might result in a different template. Therefore, introducing harmonic amplitudes as continuous variables tremendously increases the computational complexity, compared with methods with only discrete variables. Nevertheless, such hybrid models have been well studied in the literature on *switching Kalman filters* (Murphy, 1998), including effective approximations to make the computation feasible.

Improved Timing Model

Modeoing tempo (which decides note lengths) is especially meaningful when the data model is not reliable enough in some challenging cases; e.g, piano music, where many notes could happen simultaneously. Modeling tempo usually involves two issues: how to anticipate the note lengths accurately, and how to reduce computational complexity brought by the tempo variable.

Think about the beginning of Beethoven's 5^{th} symphony: "da da da dum". After hearing the first three notes "da da da", most people can anticipate the "dum" at the right moment (at the same time as in the performance). There is a reasonable assumption in practice—the tempo tends to be smooth. If it takes longer to play one note (if the performance slows down), the next note usually would take longer too. The tempo rarely jumps up and down abruptly; most of the time, it is steady, sometimes floating around slowly. We are able to predict the onset of the "dum" because we infer the tempo from the "da da da" and assume the tempo does not change at "dum". Modeling the tempo as smooth is a reasonable assumption, and will potentially help improve score-following results.

Every score-following algorithm includes discrete variables which represent the score positions. Modeling tempo leads to inference of *continuous* variables in addition to those discrete variables, dramatically increasing the computational complexity because there are infinite hypotheses for continuous variables. As mentioned before, well-established approaches exist in the switching Kalman filter literature for reducing the computation in such hybrid models.

1.5 Contributions

Piano score following is one of the most challenging cases in score following, not least because of the strong polyphonic style and the pedaling effects in piano music. This work aims to investigate potential ways to overcome the current bottleneck on piano score following, and seeks to understand the nature of the problem better by empirical experiments.

Motivated by this challenge, I propose two new methods, each designed to focus on one of the two aspects: to improve the data model by tracking the harmonic amplitudes, and to improve the timing model by assuming the tempo to be smooth. The two new methods show an interesting intersection between the score-following application and the switching Kalman filter models. One of the new methods also represents the first time a score-following algorithm has accounted for the harmonic evolution within a chord.

I have conducted experiments on 50 excerpts of real piano performance recordings consisting of 48 minutes of music, and have analyzed the results in statistical terms. The discussion based on the results includes several aspects of the score-following problem and contributes to a more nuanced understanding of this topic. I also propose a new evaluation method for the score-following problem, which is simple and straightforward, and most importantly, avoids the ambiguity in existing evaluation methods due to the trade-off between accuracy and promptness.

1.6 Organization

Chapter 2 provides the theoretical background for later chapters. It first describes how the musical score and audio are represented in a score-following program, and then gives a thorough introduction to two of the most popular state-space models: the hidden Markov model and the Kalman filter model. The hidden Markov model is also the foundation of the baseline method, which is described in this chapter as an application example. The chapter ends with an introduction to the switching Kalman filter model, which is the basis of the two proposed new methods. Chapter 3 and Chapter 4 present new methods that aim to improve respectively the data model and the timing model. Chapter 3 is also based on my previously published work, Jiang and Raphael (2019). Chapter 5 describes the experiments conducted on 50 piano excerpts, and includes an in-depth discussion of the results. This chapter also introduces a new evaluation method for the score-following problem. This text ends with Chapter 6, which concludes the work.

CHAPTER 2

Theoretical Background

2.1 Technical Background

2.1.1 Representing Scores

This section explains how a musical score is represented in computers. Monophonic music only has one musical event at a time, and therefore its score can be represented as a sequence of notes and rests, each note or rest associated with a score position, in beats. Polyphonic music can have multiple musical events happening at the same time, and those events usually form parallel lines of voices (a voice in piano music can be thought of as a sequence of note that belong to a particular "singer" in a chorus). These voices bring problems because they can no longer be represented sequentially, which is a requirement for many techniques dealing with time-series data, including the techniques involved here. To solve this, I transform the polyphonic music into a "homophonic" form—multiple voices moving at the same time with the same rhythm (Figure 2.1). This enables polyphonic music to be linearly represented in the same fashion as in monophonic music: a sequence of musical events (i.e., notes, chords, and rests), each event associated with a score position, as shown in Table 2.1. The sequence can be extracted from a MIDI file that contains the score information. For simplicity's sake, in the remainder of this text I will refer to any of these events as a "chord", unless it's necessary to draw a distinction.

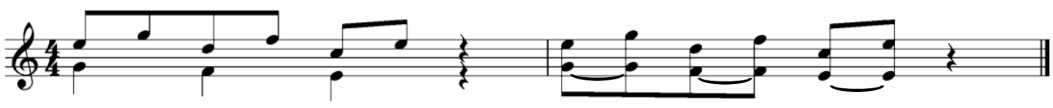


Figure 2.1: "Homophonic" view of polyphonic music (modified from Raphael (2006)). The left bar is the original score with two voices. The right bar represents this score as a sequence of chords.

Index	Onset	MIDI
1	0	{67, 76}
2	$\frac{1}{8}$	{67, 79}
3	$\frac{1}{4}$	{65, 74}
4	$\frac{3}{8}$	{67, 77}
5	$\frac{1}{2}$	{64, 72}
6	$\frac{5}{8}$	{64, 76}
7	$\frac{3}{4}$	{}

Table 2.1: Representing the score in Figure 2.1 (left bar) as a sequence of musical events. The first column is the index of the musical event; the second column is the score position of the event; the third column is the MIDI numbers (refer to Figure 1.3) involved in this event. There could be zero (a rest), one, two, or more notes in a musical event.

2.1.2 Representing Audio

The audio is read into computers in a digital form by sampling. In the experiments in Chapter 5, the sampling rate is 8 kHz (8000 samples per second). In score-following algorithms, the basic unit of the data is an audio *frame*, comprising a fixed number of samples in a row. In my experiments, for example, a frame is 512 samples long, containing 64 milliseconds of audio (512/8000 seconds). The entire audio data is evenly segmented into a sequence of frames, with some overlapping samples between adjacent frames (to make the boundary smooth). The offset between adjacent frames, also called the "hop size", is typically half of the frame length, resulting in an overlap on the other half. For example, in one of my experiments, the hop size is 256, making adjacent frames overlap with 256 samples (256 overlap = 512 frame length − 256 hop size). Figure 2.2 illustrates audio frames with the frame lengths at 20, and the hop size at 10. The goal of score-following algorithms is to decide which musical event each audio frame belongs to in an online fashion.

2.1.3 Audio Features and Data Likelihood

This section describes how to calculate the data likelihood of a frame. A frame consists of 512 consecutive samples in the audio. Researchers often transform these samples from the

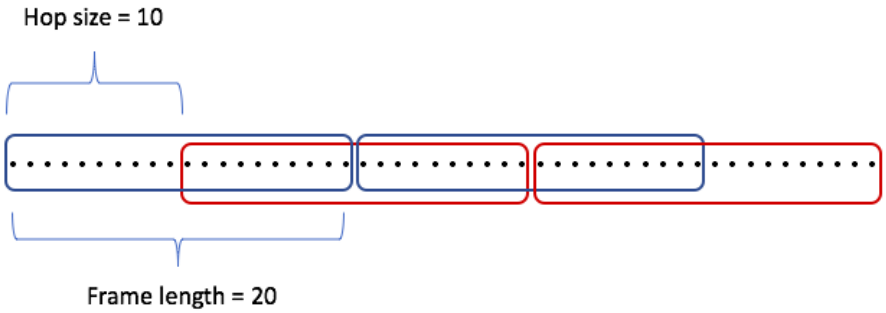


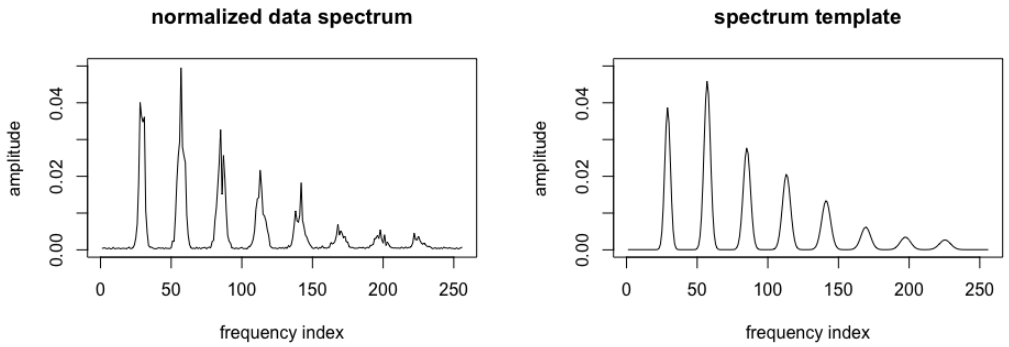
Figure 2.2: Illustration of audio frames. Each dot represents an audio sample and each rectangle represents a frame.

time domain to the frequency domain, because music signals have more intuitive features in the frequency domain (e.g., peaks at harmonic frequencies). I use the short-time Fourier transform technique (Gröchenig, 2001) to transform an audio frame to the frequency domain. We can denote a series of samples in a frame as $x_1, x_2, \ldots, x_T$, where $T = 512$. After the discrete Fourier transform, we get a sequence of complex numbers:

$$X_w = \sum_{t=1}^{T} h(t) \cdot x_t \cdot e^{-\frac{i2\pi}{T}(w-1)(t-1)}$$

where $h(t)$ is a window function, and $w = 1, \ldots, T$. Because of the periodicity of the term $e^{-\frac{i2\pi}{T}(w-1)(t-1)}$, the modulus of each element in the result is symmetrical ($|X_w| = |X_{T-w}|$), and thus it can be specified by only the first half of the sequence, which results in a vector of 256 real numbers. Each element in the vector corresponds with one of the frequencies that are distributed evenly in the range of $0 \sim 4000\ Hz$ (from 0 to half of the sampling rate). That vector is the *spectrum* of the frame, denoted as $\boldsymbol{y} = \{y_w\}$, where $y_w = |X_w|, 1 \leq w \leq W$ and $W = 256$. The index w specifies which frequency an element belongs to; y_w quantifies the intensity at the wth frequency bin. Figure 2.3a shows a spectrum observed in a frame belonging to a single note. The peaks represent this note's harmonics, as explained in Section 1.4.1. Considering that the volume of the sound should not affect the data likelihood, the features used in the calculation are the *normalized* spectrum, $\boldsymbol{y}' = \{y'_w\}$, where $y'_w = y_w / \sum_{w=1}^{W} y_w$.

Suppose there are K musical events in the musical score, and each event has a template



(a) Observed data features.

(b) The template of data features.

Figure 2.3: (a) is an example of observed data features of a single note. (b) is the template of this note. They share the same dimension, and they both sum to 1.

of its spectrum (Figure 2.3b). Denote the template of the kth event as $\boldsymbol{q}^k = \{q_w^k\}$, and it should always sum to 1, $\sum_{w=1}^{W} q_w^k = 1$. $\boldsymbol{q}^k$ represents the expected observation at each frequency (Figure 2.3b). When interpreting $\boldsymbol{q}^k$ as a probability distribution over the 256 frequency bins, and $\boldsymbol{y}'$ as the normalized "counts" of a sequence of samples independently drawn from this distribution, we can calculate the likelihood of observing the data $\boldsymbol{y}$ given the musical event k as:

$$p(\boldsymbol{y}|k) = \prod_{w=1}^{W} (q_w^k)^{y'_w} \tag{2.1}$$

This calculation resembles the multinomial distribution, but excluding the factor that doesn't change with k.

In the basic hidden Markov method (Section 2.4) and the method in Chapter 4, the template for calculating the data likelihood of a musical event is fixed: the same template is used for *all* frames of an event, no matter whether they are at the beginning, in the middle, or at the end of the event. Each note's template consists of several Gaussian shaped distributions centered at the harmonics, and their comparative heights are carefully calibrated (Figure 2.3b); a chord's template is the superposition of its notes' templates with normalization, whereas a rest's template looks like a uniform distribution. In contrast, the templates in Chapter 3 are data-driven, flexible to adapt to the changing spectrum within a note, and thus should provide a better estimation of the data.

2.2 State-space Models

State-space models are highly influenced by control engineering. In control theory, the condition of a dynamical system is abstracted by a set of variables, called "states". The state variables summarize enough information about the system at any given time for the purpose of analyzing or predicting its behavior. State-space models describe the temporal evolution of the states in a dynamical system. Because the variables change with time, state-space models are commonly used on time-series data (or sequential data in general, e.g., DNA sequences, texts). The representation of time in a system could be either continuous or discrete, but I only discuss discrete-time systems here in this text: a variable uses an integer subscript to distinguish its values at different time steps.

While the evolution of the states can be regarded as deterministic or stochastic, here I only consider the latter, because the probabilistic approach can model the *uncertainty* in a musical performance we have not seen before. Therefore, the state variables are all random variables. Usually, these states are internal and cannot be observed—hence the name "hidden states". However, hidden states can be indirectly inferred from the observable measurements of the system (the usage of the term "measurement" comes from engineering where the outputs are read from sensors). In state-space models, measurements represent the observed data (or sometimes *features* of the data). Like the hidden states, measurements are also random variables that change with time; unlike the hidden states, they are observable.

State-space models are *generative* models that can describe the process of how a data sequence is generated. They usually contain two parts: one describing how the internal states transition from one time step to the next, and the other describing how the data at each time step are generated from the corresponding state(s). Mathematically, if we write $\boldsymbol{X}_n$ as the state at time step n and $\boldsymbol{Y}_n$ as the observed data at that time, $n = 1, 2, \ldots, N$ where N is the length of the sequence, then the first part describes $P(\boldsymbol{X}_{n+1}|\boldsymbol{X}_n)$, and the second part describes $P(\boldsymbol{Y}_n|\boldsymbol{X}_n)$. The states in such models are assumed to have the Markov property: at any given time n, the future states $\boldsymbol{X}_{n+1}, \ldots, \boldsymbol{X}_N$ only depend on the current state $\boldsymbol{X}_n$, not

any of the previous states $\boldsymbol{X}_1, \boldsymbol{X}_2, \ldots, \boldsymbol{X}_{n-1}$.

The state variables are usually either all continuous or all discrete. I describe two of the most well-known state-space models: the hidden Markov model (Section 2.3) where the states are discrete, and the Kalman filter model (Section 2.5) where the states are continuous. These two also share identical dependency structures. At the end of this chapter, I introduce the *switching* Kalman filter model where there are both continuous and discrete states.

2.3 The Hidden Markov Model

2.3.1 State Graph

A hidden Markov model (HMM) always has one discrete hidden state at a time, thus the probability distribution of the state is on a finite number of values. Although we never know the exact value of the state at each time step because it is hidden and random, we do know *a priori* the nature of how the state changes (or doesn't change) with time. The state follows "soft" rules specifying which values can transition from which between neighboring time steps, each possible transition associated with a probability. These are called the transition probabilities. This information can be depicted by a state graph, as in Figure 2.4. The set of possible values for the state is called the state space, denoted as $\mathcal{S}$. In this example, there are four possible values in the state space, $\mathcal{S} : \{s_1, s_2, s_3, s_4\}$. The probabilities of the "out" arrows from a value must add up to 1. The transition probabilities can also be expressed by a square matrix, Q, where Q_{ij} is the probability of transitioning from the ith state value to the jth state value, and each row of the matrix must sum to 1. From time step n to $n+1$, the probability of the state X transitioning from value s_i to s_j is $P(X_{n+1} = s_j | X_n = s_i) = Q_{ij}$, $\forall s_i \in \mathcal{S}, s_j \in \mathcal{S}$.

At the first time step ($n = 1$), we need to specify the initial probability distribution, $\boldsymbol{\pi}$, for each possible value in $\mathcal{S}$. For example, for the case in Figure 2.4, the distribution should have four elements: $\boldsymbol{\pi} = (\pi_1, \pi_2, \pi_3, \pi_4)$ where $\sum_i \pi_i = 1$ and $\pi_i = P(X_1 = s_i), s_i \in \mathcal{S}$. Given an initial distribution on the state space, $\boldsymbol{\pi}$, and any such state graph with its transition

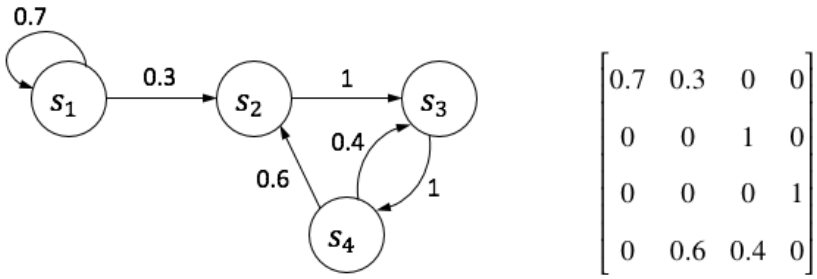


Figure 2.4: A state graph with four possible state values, and the corresponding transition matrix.

matrix Q, we can generate a random sequence $\{X_n\}, n = 1, 2, \ldots, N$, according to

$$P(X_1 = s_j) = \pi_j, \ s_j \in \mathcal{S}$$
$$P(X_{n+1} = s_j | X_n = s_i) = Q_{ij}, \ s_i, s_j \in \mathcal{S}$$

The state sequence is a homogeneous Markov chain because it has the Markov property, and the transition probabilities don't change with time:

$$P(X_{n+1}|X_1, \ldots, X_n) = P(X_{n+1}|X_n)$$

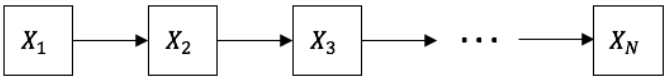


Figure 2.5: The state sequence as a Markov chain. The arrows denote conditional dependencies.

2.3.2 Dependency Graph of HMM

Although the state sequence is hidden, we can observe the *data* sequence where the data variable at each time step depends only on the value of the hidden state at that time. The observed data can be discrete or continuous. An HMM is oftentimes illustrated by a two-layer graph as in Figure 2.6. It is a special case of the dynamic Bayesian networks where the structure and parameters repeat over time (Ghahramani, 1997; Murphy and Russell, 2002).

The state graph and the initial probability distribution from Section 2.3.1 model the arrows (dependency relationships) in the top layer. The arrows toward the second layer

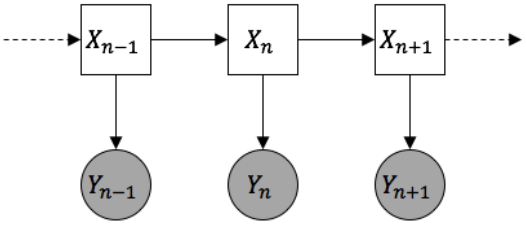


Figure 2.6: A dependency graph of an HMM. The arrows indicate the conditional dependencies among the variables. The observable variables are shaded.

indicate the "data model"; they are the probability distributions of observing the data given the state at each time step, or $P(Y_n|X_n), n = 1, \ldots, N$. From the Bayesian point of view, the former is regarded as the *prior* information for the states, because it represents available information about the states *before* observing the data; the latter is regarded as the data likelihood associated with certain state hypotheses. Through these two parts, an HMM can model the joint distribution of $P(X_1, \ldots, X_N, Y_1, \ldots, Y_N)$ recursively, and thus is a generative model:

$$P(X_1, \ldots, X_N, Y_1, \ldots, Y_N) = \underbrace{P(X_1) \prod_{n=1}^{N-1} P(X_{n+1}|X_n)}_{\text{prior}} \underbrace{\prod_{n=1}^{N} P(Y_n|X_n)}_{\text{data likelihood}}$$

2.3.3 Inference

While there are several inference tasks in HMMs (see Rabiner (1989)), here I only discuss the one that's most relevant to the score-following problem, called "filtering". The task is to compute the conditional probability distribution of the hidden state at a time step, given all the observed data from the beginning up until this time step. Writing $Y_{1:n}$ for $Y_1, \ldots, Y_n$, this conditional probability is $P(X_n|Y_{1:n}), n = 1, 2, \ldots, N$. As time goes on (or as n increases), we collect more and more data, and our knowledge about the state is "filtered" through the lens of all the available data so far. We can calculate the filtered probability for the state value s_j at time n by first calculating the joint probability $P(X_n = s_j, Y_{1:n} = y_{1:n})$, and then

$P(Y_{1:n} = y_{1:n})$:

$$P(X_n = s_j | Y_{1:n} = y_{1:n}) = P(X_n = s_j, Y_{1:n} = y_{1:n}) / P(Y_{1:n} = y_{1:n}) \tag{2.2}$$

where $P(Y_{1:n} = y_{1:n}) = \sum_{s'_j \in \mathcal{S}} P(X_n = s'_j, Y_{1:n} = y_{1:n})$. The joint probability $P(X_n = s_j, Y_{1:n} = y_{1:n})$ can be computed recursively using the forward algorithm, as described below.

Let's define $\alpha_n(s_j) = P(X_n = s_j, Y_{1:n} = y_{1:n})$. $\alpha_n(s_j)$ can be recursively computed using the transition probabilities Q, and the data likelihood:

$$\begin{aligned}
\alpha_n(s_j) &= P(X_n = s_j, Y_{1:n} = y_{1:n}) \\
&= \sum_{s_i \in \mathcal{S}} P(X_n = s_j, X_{n-1} = s_i, Y_{1:n} = y_{1:n}) \\
&= \sum_{s_i \in \mathcal{S}} P(X_{n-1} = s_i, Y_{1:n-1} = y_{1:n-1}) P(X_n = s_j | X_{n-1} = s_i) P(Y_n = y_n | X_n = s_j) \\
&= \sum_{s_i \in \mathcal{S}} \alpha_{n-1}(s_i) Q_{ij} P(Y_n = y_n | X_n = s_j), \ n = 2, 3, \ldots, N
\end{aligned}$$

The initial α is:

$$\begin{aligned}
\alpha_1(s_j) &= P(X_1 = s_j, Y_1 = y_1) \\
&= P(X_1 = s_j) P(Y_1 = y_1 | X_1 = s_j) \\
&= \pi_j P(Y_1 = y_1 | X_1 = s_j)
\end{aligned}$$

According to Equation 2.2, the filtered probability is:

$$P(X_n = s_j | Y_{1:n} = y_{1:n}) = \alpha_n(s_j) / \sum_{s'_j \in \mathcal{S}} \alpha_n(s'_j), \ n = 1, \ldots, N, \ s_j \in \mathcal{S}$$

2.4 HMM Application: Score Following

This section presents one of the state-of-the-art score-following algorithms, introduced by Raphael (1999), which uses a hidden Markov model (HMM). It was intended for monophonic music, but can easily extend to polyphonic music by adopting a homophonic view

of the score, as discussed in Section 2.1.1. The hidden state at each audio frame (Section 2.1.2) contains the information of which chord is being played at any given time, and the observation at each frame is the normalized spectrum of the audio (Section 2.1.3). The task is to determine, at each time frame, the probability distribution of the state given the observed data up until this frame. I first describe the state graph of the HMM (the prior part), and then the data model (the likelihood part), before calculating the filtered probabilities.

2.4.1 State Graph from a Musical Score

A state sequence that represents a musical performance should be consistent with the musical score: chords should appear in the same order as in the score. This constraint can be expressed by a linear state graph with ordered chords, as in Figure 2.7. Each chord is represented with a number of states, and some researchers call them "micro states". Each state has two "out" arrows: a self-loop arrow and an arrow pointing to the next state. Such a state graph can represent a variety of musical performances played with different lengths of a chord.

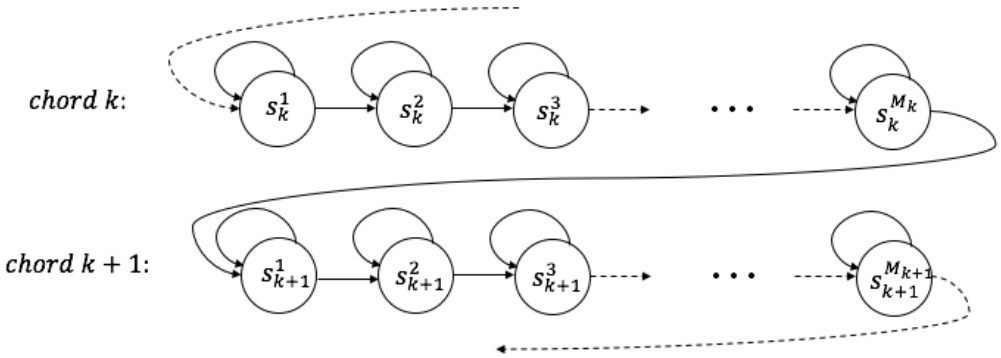


Figure 2.7: The state graph from a musical score. Chord k has M_k micro states, and chord $k+1$ has M_{k+1} micro states.

A state sequence has to go through all the micro states of a chord before entering the next chord. For the sake of simple (but sufficient) design, all states belonging to the same chord share the same self-loop probability, as illustrated in Figure 2.8. This way, the number of audio frames spent in a micro state has a geometric distribution: the probability distribution

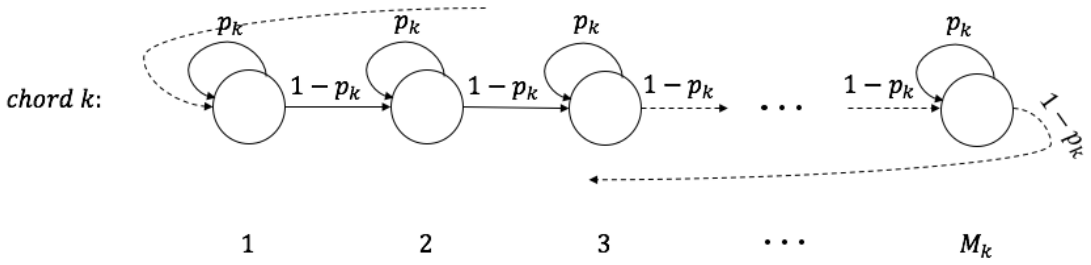


Figure 2.8: Transitions between the neighboring micro states in a chord. This chord has M_k micro states, with the self-loop probability of p_k.

of the number of trials in order to get one "success". The total number of frames spent in a *chord* is the sum of the frames spent in all its micro states. Since the sum of independent and identically distributed geometric distributions follows a negative binomial distribution, the chord length has a negative binomial distribution. For example, a chord k has M_k micro states with the self-loop probability of p_k; then, the chord length has a mean of $M_k/(1-p_k)$ and a variance of $M_k p_k/(1-p_k)^2$—we can think of this as the probability of the number of trials in order to get M_k "successes". The musical score indicates how long each chord is expected to last by its nominal duration (in beats) and a general tempo (beats per minute). Using this expected chord length as the mean, and a reasonable variance (arbitrarily set or learned from data), we can choose the best M_k and p_k to approximate the duration distribution of the kth chord. (We could also use the empirical mean and variance from a variety of performance data, if such training data are available.) If there are K chords in total, the entire state space is $\mathcal{S}: \{s_k^m\},\ 1 \le k \le K,\ 1 \le m \le M_k$. The transition probabilities are:

$$P(X_{n+1} = x_{n+1} | X_n = s_k^m) = \begin{cases} p_k, & \text{if } x_{n+1} = s_k^m \\ 1-p_k, & \text{if } x_{n+1} = s_k^{m+1},\ m < M_k \\ 1-p_k, & \text{if } x_{n+1} = s_{k+1}^1,\ m = M_k \end{cases}$$

The state graph in Figure 2.7 represents the prior information offered by the musical score, without any actual observation data. The model uses probabilities to measure the plausibility of any given state sequence that represents a version of the musical performance, by quantifying how "extreme" this sequence is from the score. In contrast to the HMM state

graph presented here, in which the chord-length distributions are always negative binomial, Cont (2010) and Cuvillier (2016) use hidden semi-Markov models (HSMMs) wherein each chord length can be modeled by *any* explicit distribution, and the transition probabilities depend on the "age" of a state (the number of frames through which the state has been sustained). Since a negative binomial distribution requires exactly two parameters, here only the mean and the variance of the chord-length distribution are modeled, and typically we do not know the shape of the distribution other than its mean and some measure of its variability anyway (Raphael, 1999). More importantly, an HMM needs fewer number of states than an HSMM, because the HMM doesn't need to keep track of the "age" of a state.

2.4.2 Data Model

The data model defines $P(Y_n = y_n | X_n = s_k^m)$, the likelihood of observing y_n if the hidden state at that frame is s_k^m—in other words, how much the observed data support a hypothesis of the state. I write this likelihood as $p(\boldsymbol{y}|s_k^m)$ in this section, because the data model doesn't vary with the frame, n, and the observation is a spectrum vector. From Equation 2.1 in Section 2.1.3, I have modeled this data likelihood as:

$$p(\boldsymbol{y}|s_k^m) = \prod_{w=1}^{W} (q_w^k)^{y'_w} \tag{2.3}$$

where $\boldsymbol{y} = \{y_w\}_{w=1}^W$, $y'_w = y_w / \sum_{w=1}^W y_w$, $W = 256$, and $\boldsymbol{q}^k$ is the template for the kth chord, $\boldsymbol{q}^k = \{q_w^k\}_{w=1}^W$. Since micro states of the same chord share the same k, they also have the same likelihood value, i.e., $p(\boldsymbol{y}|s_k^1) = p(\boldsymbol{y}|s_k^2) = \ldots = p(\boldsymbol{y}|s_k^{M_k})$.

Each event k in the musical score has one preconstructed template. If the event is a single note, the template consists of a mixture of Gaussian-shaped distributions concentrated on the harmonics of this note, as in Figure 2.3. The template of a chord is a mixture of multiple single-note templates, as demonstrated in Figure 1.4. Each template is also mixed with a small portion of background-noise distribution. Raphael (2006) provides additional details about constructing such templates.

2.4.3 Filtering

Whenever a new audio frame is received, the model adjusts its estimates of the plausibility of each state hypothesis at the current frame, considering both the prior information provided by the musical score, and how the evidence (i.e., observed data) supports each hypothesis. The calculation of the filtered probabilities in general is described in Section 2.3.3. Below, I show the calculation of the filtered probability, $P(X_n = s_k^m | Y_{1:n} = \boldsymbol{y}_{1:n})$, for this score-following application, which follows the same steps as in the general case.

According to the state graph, a state s_k^m at frame n has two predecessors: the state itself, and the state immediately preceding it (s_k^{m-1} or $s_{k-1}^{M_{k-1}}$, depending on whether s_k^m is the first micro state of the kth chord). When $m > 1$, the filtered probability is:

$$
\begin{aligned}
& P(X_n = s_k^m | Y_{1:n} = \boldsymbol{y}_{1:n}) \\
= \; & \frac{1}{Z_n} \Big[P(X_{n-1} = s_k^m | Y_{1:n-1} = \boldsymbol{y}_{1:n-1}) P(X_n = s_k^m | X_{n-1} = s_k^m) P(Y_n = \boldsymbol{y}_n | X_n = s_k^m) + \\
& \quad P(X_{n-1} = s_k^{m-1} | Y_{1:n-1} = \boldsymbol{y}_{1:n-1}) P(X_n = s_k^m | X_{n-1} = s_k^{m-1}) P(Y_n = \boldsymbol{y}_n | X_n = s_k^m) \Big] \\
= \; & \frac{1}{Z_n} P(Y_n = \boldsymbol{y}_n | X_n = s_k^m) \Big[p_k P(X_{n-1} = s_k^m | Y_{1:n-1} = \boldsymbol{y}_{1:n-1}) + \\
& \quad (1 - p_k) P(X_{n-1} = s_k^{m-1} | Y_{1:n-1} = \boldsymbol{y}_{1:n-1}) \Big]
\end{aligned}
$$

where $Z_n = \sum_{s \in \mathcal{S}} P(X_n = s | Y_{1:n} = \boldsymbol{y}_{1:n})$. If we write $p(s | \boldsymbol{y}_{1:n}) = P(X_n = s | Y_{1:n} = \boldsymbol{y}_{1:n})$ and $p(\boldsymbol{y}_n | s_k^m) = P(Y_n = \boldsymbol{y}_n | X_n = s_k^m)$, the above equation becomes:

$$
p(s_k^m | \boldsymbol{y}_{1:n}) = \frac{1}{Z_n} p(\boldsymbol{y}_n | s_k^m) \big[p_k p(s_k^m | \boldsymbol{y}_{1:n-1}) + (1 - p_k) p(s_k^{m-1} | \boldsymbol{y}_{1:n-1}) \big]
$$

For the other case of $m = 1$, the filtered probability has similar factors, just replacing s_k^{m-1} with $s_{k-1}^{M_{k-1}}$, and $(1 - p_k)$ with $(1 - p_{k-1})$. Putting the two cases together, we have:

$$
p(s_k^m | \boldsymbol{y}_{1:n}) = \begin{cases} \frac{1}{Z_n} p(\boldsymbol{y}_n | s_k^m) \big[p_k p(s_k^m | \boldsymbol{y}_{1:n-1}) + (1 - p_k) p(s_k^{m-1} | \boldsymbol{y}_{1:n-1}) \big], & m > 1 \\ \frac{1}{Z_n} p(\boldsymbol{y}_n | s_k^m) \big[p_k p(s_k^m | \boldsymbol{y}_{1:n-1}) + (1 - p_{k-1}) p(s_{k-1}^{M_{k-1}} | \boldsymbol{y}_{1:n-1}) \big], & m = 1 \end{cases}
$$

which can be calculated iteratively at each frame.

The above shows how to calculate the filtered probability iteratively as the audio frame

index increases from 1 to N. However, the number of hypotheses still increase linearly with n. It's common to use the *beam search* technique to control the size of the hypotheses: at each frame, keep no more than a fixed number of hypotheses with the highest filtered probabilities, and prune out the rest.

2.5 The Kalman Filter Model

The variables in a Kalman filter model are always continuous, and they are all normally distributed. The relationship among these variables is linear. For these reasons, the Kalman filter model is also referred to as the linear dynamical system, or the linear Gaussian state-space model. The Kalman filter algorithm has been widely used in object tracking, signal processing, econometrics, and beyond. In a nutshell, Kalman filtering is a method for filtering out the "noise" from the measurements, so we can get more accurate estimates about the "signal" of interest.

The Kalman filter can have a variety of forms, and I use a simple but typical one:

$$y_n = h_n x_n + v_n \tag{2.4}$$

$$x_{n+1} = f_n x_n + w_n \tag{2.5}$$

$$x_1 \sim N(\mu_{x,1}, \sigma^2_{x,1}) \tag{2.6}$$

$$v_n \sim N(0, \sigma^2_{v,n}),\ n = 1, \ldots, N \tag{2.7}$$

$$w_n \sim N(0, \sigma^2_{w,n}),\ n = 1, \ldots, N \tag{2.8}$$

The v_n's and w_n's are all mutually independent, and are independent from x_1. The internal states are $\{x_n\}$ and the measurements are $\{y_n\}$, $n = 1, \ldots, N$ (in lowercase for the sake of convention, adjusted from the previous usage in this chapter). f_n and h_n are known constants that model the transition of the state, and the relationship between the measurement and the state, respectively. w_n and v_n are random variables that model the noise involved in the state transition and the noise in the measurement. They both have 0-mean Gaussian

distributions. This Kalman filter has a structure identical to the HMM, as in Figure 2.9.

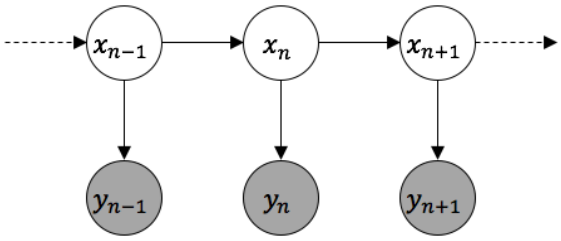


Figure 2.9: The dependency graph of the Kalman filter model in Equations 2.4-2.8. All variables are continuous. The observable variables are shaded.

The goal is to calculate the filtered probability $p(x_n|y_{1:n})$, $n = 1, 2, \ldots, N$, as the system receives more and more data. Because all random variables are jointly normally distributed, distributions conditioned on any set of these variables are also normal. Denote the mean of such a conditional distribution as a function $\mu_{V_1}(V_2)$ where V_1 is the name of the random variable, and V_2 is the condition; similarly, we have the function $\sigma^2_{V_1}(V_2)$ as the corresponding variance. For example, the mean and variance of the filtered probability are $\mu_{x_n}(y_{1:n})$ and $\sigma^2_{x_n}(y_{1:n})$.

The Kalman filter algorithm has two iterative steps: first, it uses the filtered distribution at the current time step to *predict* the state at the next time step before observing the data; second, it *updates* the filtered distribution at the next time step considering the newly observed data. These steps are expressed in the following formulas:

$$
\begin{aligned}
\textit{predict} \begin{cases} \mu_{x_{n+1}}(y_{1:n}) = f_n \mu_{x_n}(y_{1:n}) \\ \sigma^2_{x_{n+1}}(y_{1:n}) = f_n^2 \sigma^2_{x_n}(y_{1:n}) + \sigma^2_{w,n} \end{cases} \\
\textit{update} \begin{cases} \mu_{x_{n+1}}(y_{1:n+1}) = \mu_{x_{n+1}}(y_{1:n}) + k_{n+1}(y_{n+1} - h_{n+1}\mu_{x_{n+1}}(y_{1:n})) \\ \sigma^2_{x_{n+1}}(y_{1:n+1}) = (1 - k_{n+1}h_{n+1})\sigma^2_{x_{n+1}}(y_{1:n}) \end{cases}
\end{aligned}
$$

where $k_{n+1} = \frac{h_{n+1}\sigma^2_{x_{n+1}}(y_{1:n})}{h^2_{n+1}\sigma^2_{x_{n+1}}(y_{1:n}) + \sigma^2_{v,n+1}}$, which is called the Kalman gain. The Kalman filter guarantees that the filtered state estimation (which is a function of $y_{1:n}$) has a minimum mean-square error over all functions of $y_{1:n}$:

$$
\mu_{x_n}(y_{1:n}) = \arg\min_f E\left[\left(x_n - f(y_{1:n})\right)^2\right]
$$

Since we are not predicting anything in the score-following application, we can combine the two steps into one for updating the new filtered probability from the filtered probability at the previous time step:

$$\begin{aligned}\mu_{x_{n+1}}(y_{1:n+1}) &= f_n \mu_{x_n}(y_{1:n}) + k_{n+1}(y_{n+1} - h_{n+1} f_n \mu_{x_n}(y_{1:n})) \\ \sigma^2_{x_{n+1}}(y_{1:n+1}) &= (1 - k_{n+1} h_{n+1})(f_n^2 \sigma^2_{x_n}(y_{1:n}) + \sigma^2_{w,n})\end{aligned} \tag{2.9}$$

The Kalman filter model is a generative model that describes the process of generating the measurement data. It has two iterative steps: first, we sample the state at time n from the normal density function $N(x_n; f_{n-1}x_{n-1}, \sigma^2_{w,n-1})$; second, we sample the measurement at time n from $N(y_n; h_n x_n, \sigma^2_{v,n})$. Sometimes, it's useful to marginalize out the state variables and compute the marginal likelihood for the observed data (used in Section 4.1.1). The recursive calculation is:

$$\begin{aligned}p(y_{1:N}) &= p(y_1) \prod_{n=1}^{N-1} p(y_{n+1}|y_{1:n}) \\ &= p(y_1) \prod_{n=1}^{N-1} \int_{x_{n+1}} p(y_{n+1}|x_{n+1}) p(x_{n+1}|y_{1:n}) \, \mathrm{d}x_{n+1} \\ &= p(y_1) \prod_{n=1}^{N-1} \int_{x_{n+1}} N(y_{n+1}; h_{n+1}x_{n+1}, \sigma^2_{v,n+1}) N(x_{n+1}; \mu_{x_{n+1}}(y_{1:n}), \sigma^2_{x_{n+1}}(y_{1:n})) \, \mathrm{d}x_{n+1} \\ &= p(y_1) \prod_{n=1}^{N-1} N(y_{n+1}; h_{n+1}\mu_{x_{n+1}}(y_{1:n}), \sigma^2_{v,n+1} + h^2_{n+1}\sigma^2_{x_{n+1}}(y_{1:n}))\end{aligned}$$

For other forms of the Kalman filter, there might be a known control-input term in Equation 2.5 (see Welch et al. (1995)), or an extra but known term in Equation 2.4 (see Chapter 4), but the "predict" and "update" formulas are very similar. Treating the random variables as scalars is sufficient for the purpose of this text. However, more generally, the random variables are oftentimes treated as vectors instead of scalars, as in Durbin and Koopman (2012).

2.6 Switching Kalman Filters

Switching Kalman filters (SKFs), sometimes called switching state-space models, have both discrete and continuous state variables. The "switching" refers to the transitioning of the discrete state in adjacent time steps. The discrete states are used to switch among different Kalman filter models. Given the result of the "switching" at a time step, the transitioning of the continuous state and the relationship between the state and the measurement are still linear (just like in Kalman filter models), but the parameters depend on the switched discrete state. The "switching" in SKFs allows for approximations to non-linear and non-Gaussian models, extending the capability of the Kalman filter models, which can only represent linear Gaussian models.

2.6.1 An Example Structure

SKFs can have a variety of forms, and here I present what Murphy (1998) calls the "canonical" form, as in Figure 2.10. Denote the discrete state process as $\{s_n\}_{n=1}^N$ and the continuous state process as $\{x_n\}_{n=1}^N$. The discrete state has a state space of $\mathcal{S}$, and its process is a Markov chain which is described in Section 2.3.1. The parameters for the continuous state and the measurement data are dependent on the discrete state, so we replace them with functions of the discrete state in Equations 2.4-2.8:

$$y_n = h(s_n)\, x_n + v_n \tag{2.10}$$

$$x_{n+1} = f(s_{n+1})\, x_n + w_n \tag{2.11}$$

$$x_1 \sim N(\mu_{x,1}, \sigma_{x,1}^2) \tag{2.12}$$

$$v_n \sim N(0, \sigma_v^2(s_n)),\ n = 1, \ldots, N \tag{2.13}$$

$$w_n \sim N(0, \sigma_w^2(s_n)),\ n = 1, \ldots, N \tag{2.14}$$

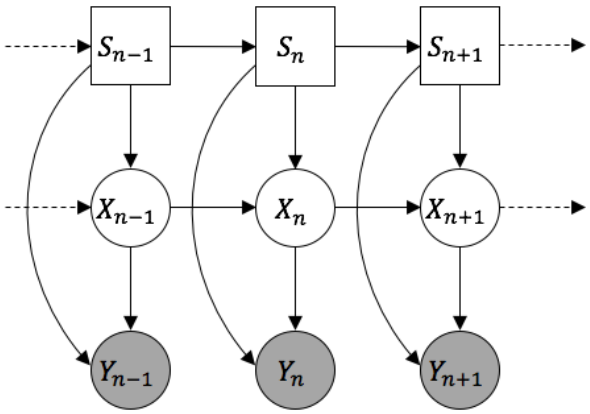


Figure 2.10: The dependency graph of a switching Kalman filter. The circles are continuous variables and the squares are discrete variables. The observable variables are shaded.

The SKF is also a generative model:

$$p(s_{1:N}, x_{1:N}, y_{1:N}) = \underbrace{p(s_1)p(x_1|s_1)\prod_{n=1}^{N-1} p(s_{n+1}|s_n)p(x_{n+1}|s_{n+1}, x_n)}_{\text{prior}} \underbrace{\prod_{n=1}^{N} p(y_n|s_n, x_n)}_{\text{data likelihood}}$$

When $\{s_n\}_{n=1}^N$ is fixed, Equations 2.10-2.14 represent a simple Kalman filter model described in Section 2.5. However, $\{s_n\}_{n=1}^N$ is a random process that we cannot observe, and the number of configurations grows exponentially when n increases from 1 to N, each configuration corresponding with a different Kalman filter model.

2.6.2 Computation

At every time step, we want to calculate the filtered joint distribution of the discrete and the continuous states, $p(s_n, x_n|y_{1:n}), n = 1, \ldots, N$. Denote $\mathcal{P}(s_n)$ as the set of all paths of the discrete state that end with s_n. For a fixed path $s_{1:n} \in \mathcal{P}(s_n)$, we can easily calculate the corresponding filtered distribution (a Gaussian) for the continuous state x_n according to Equation 2.9. Since the calculation also depends on $s_{1:n}$, we write the mean and variance of this distribution as $\mu_{x_n}(y_{1:n}, s_{1:n})$ and $\sigma^2_{x_n}(y_{1:n}, s_{1:n})$. The filtered joint distribution is the summation of the probabilities for possible history paths leading to state s_n; it can also be

viewed as summing out the $s_{1:n-1}$ in $p(s_{1:n-1}, s_n, x_n|Y_{1:n})$:

$$\begin{aligned} p(s_n, x_n|y_{1:n}) &= \sum_{s_{1:n}\in\mathcal{P}(s_n)} p(s_{1:n}, x_n|y_{1:n}) \\ &= \sum_{s_{1:n}\in\mathcal{P}(s_n)} p(s_{1:n}|y_{1:n}) p(x_n|y_{1:n}, s_{1:n}) \\ &= \sum_{s_{1:n}\in\mathcal{P}(s_n)} p(s_{1:n}|y_{1:n}) N(x_n; \mu_{x_n}(y_{1:n}, s_{1:n}), \sigma^2_{x_n}(y_{1:n}, s_{1:n})) \end{aligned} \tag{2.15}$$

This is a scaled Gaussian mixture distribution, with the scale being

$$p(s_n|y_{1:n}) = \sum_{s_{1:n}\in\mathcal{P}(s_n)} p(s_{1:n}|y_{1:n})$$

The number of components grows exponentially with n—obviously, the computation is intractable without approximation.

Moment Matching

Although there are a variety of approximation methods to limit the number of components in the Gaussian mixture (see Murphy (1998)), I focus on *moment matching*, which collapses multiple Gaussians into a single Gaussian. Using this method, the resulting Gaussian has the same mean and variance (the first two moments of a distribution) as the original mixture. For example, say there are R components in the mixture: $\{g_r N(\mu_r, \sigma_r^2)\}_{r=1}^R$ where g_r's are the weights, and they add to 1. Then, the replacement Gaussian is $N(\mu, \sigma^2)$ where:

$$\begin{aligned} \mu &= \frac{1}{g}\sum_{r=1}^{R} g_r \mu_r \\ \sigma^2 &= \frac{1}{g}\sum g_r[\sigma_r^2 + (\mu_r - \mu)^2] \\ g &= \sum_{r=1}^{R} g_r = 1 \end{aligned}$$

This is proven to be the optimal approximation in the sense of Kullback-Leibler distance (Runnalls, 2007). Mathematically, the g_r's do not have to add to 1. In that case, g is the scale of the Gaussian mixture distribution. Therefore, the scaled Gaussian mixture distribution

in Equation 2.15 can be approximated as a scaled Gaussian distribution:

$$p(s_n, x_n|y_{1:n}) \approx g(y_{1:n}, s_n)N(x_n; \mu_{x_n}(y_{1:n}, s_n), \sigma^2_{x_n}(y_{1:n}, s_n)) \tag{2.16}$$

Notice that the scale, mean, and variance all depend on $y_{1:n}$ and s_n. The joint filtered distribution expressed in Equation 2.16 can be calculated iteratively, with the help of the moment matching technique.

Recursive Calculation

If we write $W_{n+1} = p(y_{n+1}|y_{1:n})$ and define $\mathcal{S}(s_{n+1})$ as the set of predecessor states of s_{n+1}, then the joint filtered distribution can be written as:

$$\begin{aligned}
&p(s_{n+1}, x_{n+1}|y_{1:n+1}) = \frac{1}{W_{n+1}} p(s_{n+1}, x_{n+1}, y_{n+1}|y_{1:n}) \\
=&\frac{1}{W_{n+1}} \sum_{s_n \in \mathcal{S}(s_{n+1})} \int_{x_n} p(s_n, x_n, s_{n+1}, x_{n+1}, y_{n+1}|y_{1:n})\, \mathrm{d}x_n \\
=&\frac{1}{W_{n+1}} \sum_{s_n \in \mathcal{S}(s_{n+1})} \int_{x_n} p(s_n, x_n|y_{1:n}) p(s_{n+1}|s_n) p(x_{n+1}|x_n, s_{n+1}) p(y_{n+1}|x_{n+1}, s_{n+1})\, \mathrm{d}x_n
\end{aligned}$$

Replacing $p(s_n, x_n|y_{1:n})$ with the approximated scaled Gaussian and putting factors that don't change with x_n outside of the integral, the above expression becomes:

$$\begin{aligned}
&\frac{1}{W_{n+1}} \sum_{s_n \in \mathcal{S}(s_{n+1})} g(y_{1:n}, s_n) p(s_{n+1}|s_n) p(y_{n+1}|x_{n+1}, s_{n+1}) \cdot \\
&\qquad \int_{x_n} N(x_n; \mu_{x_n}(y_{1:n}, s_n), \sigma^2_{x_n}(y_{1:n}, s_n)) p(x_{n+1}|x_n, s_{n+1})\, \mathrm{d}x_n
\end{aligned}$$

Using Equation 2.11, the integral part equals $N(x_{n+1}; f(s_{n+1})\mu_{x_n}(y_{1:n}, s_n), \sigma^2_{x_n}(y_{1:n}, s_n) + \sigma^2_w(s_{n+1}))$. Then, applying Equation 2.10, we have

$$\begin{aligned}
&p(s_{n+1}, x_{n+1}|y_{1:n+1}) \\
=&\frac{1}{W_{n+1}} \sum_{s_n \in \mathcal{S}(s_{n+1})} g(y_{1:n}, s_n) p(s_{n+1}|s_n) \cdot \\
&\qquad N(y_{n+1}; h(s_{n+1}) f(s_{n+1}) \mu_{x_n}(y_{1:n}, s_n), \sigma^2_{x_n}(y_{1:n}, s_n) + \sigma^2_w(s_{n+1}) + \sigma^2_v(s_{n+1}))
\end{aligned}$$

which is a scaled Gaussian mixture distribution. We know $g(y_{1:n}, s_n)$, $\mu_{x_n}(y_{1:n}, s_n)$, and $\sigma^2_{x_n}(y_{1:n}, s_n)$ from the previous iteration, and the rest of the terms in the summation are provided from the model. Define

$$g'(y_{1:n+1}, s_{n+1})N(x_{n+1}; \mu_{x_{n+1}}(y_{1:n+1}, s_{n+1}), \sigma^2_{x_{n+1}}(y_{1:n+1}, s_{n+1})) \tag{2.17}$$

as the approximated scaled Gaussian for the summation term, and we have

$$p(s_{n+1}, x_{n+1}|y_{1:n+1}) \approx \frac{1}{W_{n+1}} g'(y_{1:n+1}, s_{n+1})N(x_{n+1}; \mu_{x_{n+1}}(y_{1:n+1}, s_{n+1}), \sigma^2_{x_{n+1}}(y_{1:n+1}, s_{n+1}))$$

Using Equation 2.17 as the approximation, the normalization term becomes

$$W_{n+1} = \sum_{s_{n+1}} \int_{x_{n+1}} p(s_{n+1}, x_{n+1}, y_{n+1}|y_{1:n}) \, \mathrm{d}x_{n+1} = \sum_{s_{n+1}} g'(y_{1:n+1}, s_{n+1})$$

Therefore,

$$p(s_{n+1}, x_{n+1}|y_{1:n+1}) \approx g(y_{1:n+1}, s_{n+1})N(x_{n+1}; \mu_{x_{n+1}}(y_{1:n+1}, s_{n+1}), \sigma^2_{x_{n+1}}(y_{1:n+1}, s_{n+1}))$$

where $g(y_{1:n+1}, s_{n+1}) = \frac{g'(y_{1:n+1}, s_{n+1})}{\sum_{s_{n+1}} g'(y_{1:n+1}, s_{n+1})}$.

For more discussions about switching Kalman filters, please refer to Murphy (1998), Kim (1994), and Ghahramani and Hinton (2000).

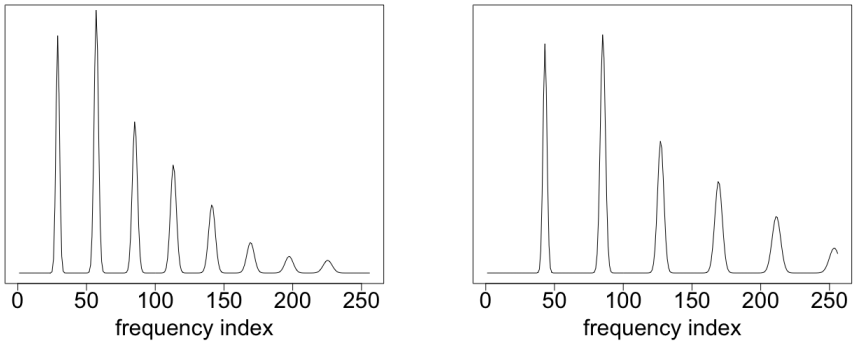
CHAPTER 3

Modeling Timbre

3.1 Motivation

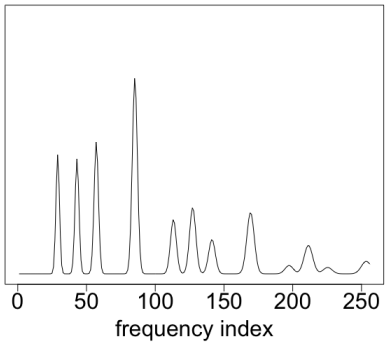
In the HMM method (Section 2.4), a template used in the data model is a mixture of all notes involved in a chord, and each note is composed of a Gaussian mixture, one component for each harmonic of the note. For example, Figure 3.1 shows the template of a single note A4 or E5, along with two possible templates of the chord "A4 and E5". The template for each chord is carefully calibrated, but *fixed*—it cannot adapt to the given data once it is (pre)defined. In other words, the HMM method assumes that the spectrum of a chord is known before observing the data, and that it does not change over the lifetime of the chord (from the onset frame until the onset of the next chord). However, in fact, we do not know the relative ratio of the notes in a chord or the true templates for the individual notes beforehand, thus cannot accurately anticipate the template of this chord (e.g., the two lower panels in Figure 3.1), and the chord's spectrum *does* evolve over time. Strictly speaking, the "harmonics" should be referred to as "partials" because piano exhibits *inharmonicity* in some higher frequencies: those spectral peaks are close to the multiples of the fundamental frequency, but not the exact values. Therefore, I choose to use the term "partials" throughout this chapter.

The piano is a percussion instrument; the sound of each note decays over time, in sharp contrast to instruments like the violin where the entire evolution of each note remains under the player's control. The rate of decay differs among different partials, with higher frequency partials usually decaying faster than the lower ones, thus leading to a *changing* spectrum in the same chord. The basic HMM method, however, cannot capture this phenomenon, because all frames within a chord share the same fixed spectrum template. This causes

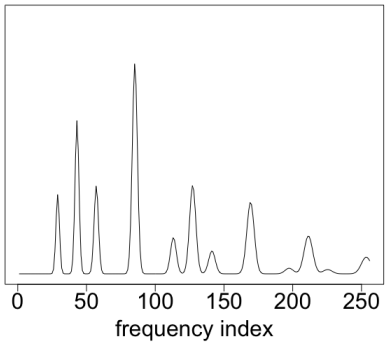


(a) Note A4's spectrum profile.

(b) Note E5's spectrum profile.



(c) The spectrum profile of A4 mixed with E5 by the ratio of 1:1.



(d) The spectrum profile of A4 mixed with E5 by the ratio of 1:2.

Figure 3.1: (a) and (b) are the spectrum profiles of two different notes. (c) and (d) are the spectrum profiles of a two-note chord, played with two different relative loudness ratios.

a mismatch between the data templates and the real observed data. As an alternative, I propose a new method that uses *flexible* templates, allowing them to adapt to the changing energy distribution among harmonics during a chord's lifetime. This new method updates the spectrum template at every frame after observing the newly received data. Because partials' relative intensities represent the timbre aspect of a musical sound, this method can be regarded as score-following by tracking the timbre changes within every chord—the *timbre tracking method*.

3.2 Assumptions

Given a specific chord in the score, we know *where* its partials lie in the frequency domain: the partials of a chord include the first harmonic of every note in the chord, and each note's higher partials (nearly multiples of the fundamental frequency; I have considered the inharmonicity of a piano when deciding higher partials' frequencies). We cannot, however, be sure about these partials' relative intensities in the sound, so their amplitudes are regarded as variables in the model. Every partial in the frequency domain is modeled with the shape of a truncated Gaussian density function, as in the left column of Figure 3.2. Some of these partials might overlap in frequency; it is common for harmonics from different notes to "collide" at the same frequency, creating an *identifiability* problem in distinguishing their amplitudes. In practice, I address this by merging them, treating them collectively as a single partial that also has the shape of a truncated Gaussian, and with the same frequency boundary as the group (as shown in the middle column of Figure 3.2). This merging procedure is done for the sake of computational tractability.

For a chord, its spectrum is divided into multiple frequency regions according to the locations of the (merged) partials, so that each region is completely nonoverlapping from the others (by the dotted line in the figure). To make the computation feasible, I assume that these partials have *independent trajectories*, and each amplitude can be independently tracked by a Kalman filter. This assumption will be made more explicit in the next section.

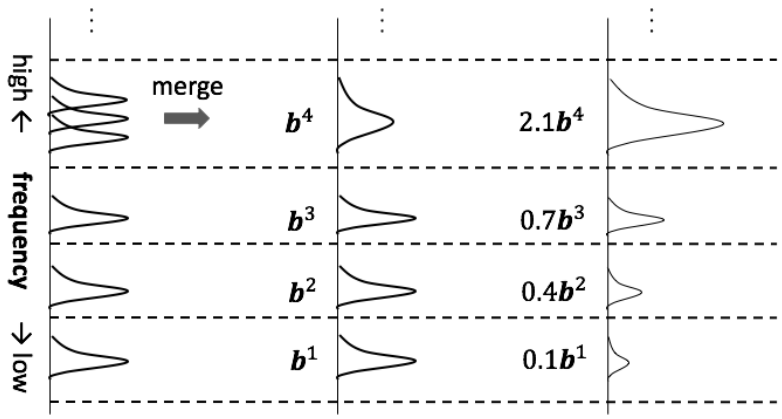


Figure 3.2: Demonstration of partials. Left: the original structure of the first six partials; Middle: the partial structure of four independent partials after merging; Right: four partials with different amplitudes. Each region divided by the dotted lines corresponds to one independent Kalman filter.

The right column in Figure 3.2 represents the partials with different amplitudes.

3.3 The Model

I first describe how a Kalman filter tracks a single partial of a chord, before explaining how the Kalman filter fits into the HMM framework, resulting in a *switching* Kalman filter model that tracks the changing features during the evolution of a chord.

3.3.1 Tracking One Partial Using a Kalman Filter

The amplitude of each partial is tracked by a Kalman filter. The partials have non-overlapping support, and I assume these Kalman filters evolve independently from each other. This section describes how a single Kalman filter tracks the amplitude of one partial (including merged partials) over the lifetime of a chord. Let's look at the pth partial of a chord, $p = 1, \ldots, P$ where P is the total number of partials in this chord. The shape of this partial is denoted by $\boldsymbol{b}^p$, which is truncated within a limited range of frequencies, as in the left column of Figure 3.3. $\boldsymbol{b}^p$ is a constant vector with the norm of 1:

$$\|\boldsymbol{b}^p\| = \left(\sum_{w=l(p)}^{h(p)} (b_w^p)^2 \right)^{1/2} = 1$$

where $l(p)$ and $h(p)$ are the lowest and highest spectrum indices of this partial.

Denote i and j as the boundary frames of this chord, and this partial's amplitude at frame n as a_n^p, $i \leq n \leq j$. At the onset frame, i, the initial amplitude of a partial has a default Gaussian distribution of $N(m_0, v_0)$, where m_0 and v_0 are the initial mean and variance. The amplitude decays with time, decaying exponentially at rate λ (< 1). This decay is modeled as

$$a_n^p = \lambda\, a_{n-1}^p + \epsilon_n^p \tag{3.1}$$

where $\epsilon_n^p \sim N(0, \sigma^2)$. The ϵ_n^p's are mutually independent over p and n. Figure 3.3 shows the decay of a partial over three frames. Note that we are *not* tracking the amplitude of every frequency, but the amplitude of $\boldsymbol{b}^p$.

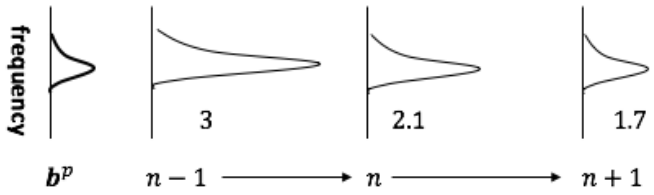


Figure 3.3: The shape of a partial (left), and its amplitude's decay between neighboring frames.

Let's denote the observed spectrum at the nth frame as $\boldsymbol{y}_n$. The observation *within the frequency range* of the pth partial is modeled as

$$\boldsymbol{y}_n^p = a_n^p \boldsymbol{b}^p + \boldsymbol{\delta}_n^p \tag{3.2}$$

where the values in $\boldsymbol{\delta}_n^p$ are independent 0-mean Gaussian noise, $\boldsymbol{\delta}_n^p \sim N(\boldsymbol{0}, \rho^2 I)$. Under these assumptions, Equations 3.1 and 3.2 form a standard Kalman filter model. As discussed in Section 3.2, observations outside the frequency range of $\boldsymbol{b}^p$ are assumed to be independent from this Kalman filter's process; therefore, we can ignore them in the current discussion. Our goal is to compute the filtered probability of the amplitude at a frame, given all the previously observed data (within this chord), $p(a_n^p | \boldsymbol{y}_i^p, \ldots, \boldsymbol{y}_n^p)$ where i is the starting frame of this chord. From the discussion in Section 2.5, we know that this filtered probability has

a Gaussian distribution, and it can be written as

$$a_n^p | \boldsymbol{y}_{i:n}^p \sim N(m_n^p, v_n^p)$$

The Kalman filter provides a straightforward update procedure (see Equation 2.9) to compute m_n^p and v_n^p as n increases from i to j:

$$\begin{aligned} m_{n+1} &= \lambda m_n + \frac{s_n}{\rho^2 + s_n \|\boldsymbol{b}^p\|^2}((\boldsymbol{b}^p)^T \boldsymbol{y}_n^p - \lambda m_n \|\boldsymbol{b}^p\|^2) \\ &= \lambda m_n + \frac{s_n}{\rho^2 + s_n}((\boldsymbol{b}^p)^T \boldsymbol{y}_n^p - \lambda m_n) \qquad (3.3) \\ v_{n+1} &= s_n - \frac{s_n^2 \|\boldsymbol{b}^p\|^2}{\rho^2 + s_n \|\boldsymbol{b}^p\|^2} \\ &= s_n - \frac{s_n^2}{\rho^2 + s_n} \qquad (3.4) \end{aligned}$$

where $s_n = \lambda^2 v_n + \sigma^2$ and $\|\boldsymbol{b}^p\| = 1$. Note that the term $(\boldsymbol{b}^p)^T \boldsymbol{y}_n^p$ in the first formula has intuitive meaning: it is the scalar projection of the observed data, $\boldsymbol{y}_n^p$, onto the shape of the partial, $\boldsymbol{b}^p$ (which is a unit vector). Therefore, the real "observed amplitude" for this partial is the scalar projection, $(\boldsymbol{b}^p)^T \boldsymbol{y}_n^p$. This makes sure that observed peaks not exactly centered at the right frequency are not fully incorporated into this partial, as demonstrated in Figure 3.4.

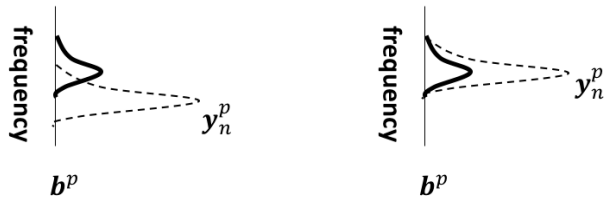


Figure 3.4: Scalar projection scenarios.

The discussion above addresses the evolution of a single partial, tracked by one Kalman filter. As mentioned in the previous section, I assume all P partials evolve independently, and each is tracked by a Kalman filter. This independence assumption is justified, because the Kalman update procedure concerning a partial's amplitude only involves the part of the data that's within the spectrum region of this partial. This means that these Kalman filters'

noise *processes* are conditionally independent from each other given the data, while the noises within a single Kalman filter at different frames are *not* conditionally independent given the data.

3.3.2 A Switching Kalman Filter

This section describes how the Kalman filters for tracking individual partials extend the filtering framework of the HMM model discussed in Section 2.4. In Figure 3.5, the discrete hidden state, X, still follows the state graph in Figure 2.7, which consists of a chain of micro states with self-loops. That is to say, the discrete transition probability, $p(x_{n+1}|x_n)$, is the same as before. The newly added middle layer (cf. Figure 2.6), notated as A, represents the amplitude information of *all* partials in state X's corresponding chord. The arrows pointing towards the observed data, Y, indicate that the observed data depend not only on the chord index, but also on the tracked amplitudes at that time.

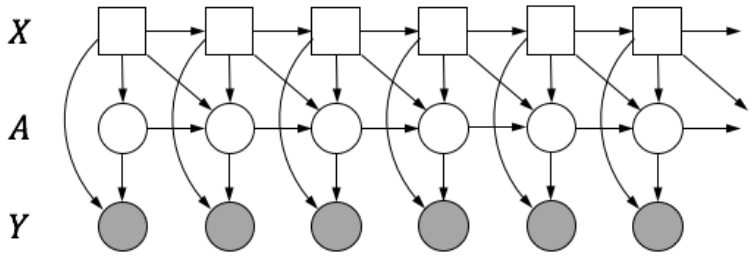


Figure 3.5: Directed acyclic graph showing the conditional independence structure of the model variables. X and A are hidden variables, and Y is observable variable (shaded). The circles are continuous variables and the squares are discrete variables.

From frame n to $n+1$, the amplitudes evolve in two different styles depending on the values of the new state, x_{n+1}. The new state either remains in the same chord, $C(x_{n+1}) = C(x_n)$, or must move to the subsequent chord, $C(x_{n+1}) = C(x_n)+1$, where $C(\cdot)$ is a function that returns the corresponding chord index of a given (micro) state. In the former case, the structure of the partials remains the same ($P(x_n) = P(x_{n+1})$, where $P(x_n)$ is the number of partials in chord $C(x_n)$), and each partial simply follows the evolution process described in the Kalman filter model in Section 3.3.1. In the latter case, a new chord is starting, with a

different partial structure from the previous chord. Some of these partials are newly formed and might be located at different frequencies from the previous chord.

Since we know almost nothing about the amplitudes of these partials at an onset frame, we may assume a default distribution for them, $a^p_{n+1}(x_{n+1}) \sim N(m_0, v_0), p = 1, \ldots, P(x_{n+1})$, where m_0 and v_0 are the initial mean and variance of a partial's amplitude. On the other hand, some partials in this new chord might be *continuing* partials from the previous chord. For example, at the onset frame of the second chord in Figure 3.6, although a new chord with a different partial structure is emerging, the *first* partial of the lower E note (fundamental frequency 330 Hz) should not be treated as newly formed: the tie indicates that this note is held throughout this chord change, and thus this partial is not reset by the player but continues its decaying process. Therefore, it is more reasonable to assume continuing partials like this simply evolve according to the Kalman filter model in Section 3.3.1, even during chord transitions. Sometimes, a continuing partial will collide with a newly formed partial in frequency. For example, the second partial of the lower E (at frequency 660 Hz) from the first chord continues during the chord transition because of the held note; however, it collides with the newly formed first partial of the E note an octave above at the same frequency. In cases like this, we should treat the collided partial as a newly formed partial with the default distribution $N(m_0, v_0)$, because we cannot predict the energy contribution brought by the new note. Figure 3.6 illustrates the cases of continuing partials and collided partials.

Let's look at the *data likelihood* calculation, in the case of a continuing chord, $C(x_{n+1}) = C(x_n)$. According to Equation 3.2, given the hypothesis state x_{n+1}, the data at frame $n+1$ depends on the tracked amplitudes of the partials at this frame. Say the corresponding chord of this state, $C(x_{n+1})$, has multiple partials, with amplitudes $\{a^p_{n+1}(x_{n+1})\}$, $p = 1, \ldots, P(x_{n+1})$, each of which is tracked by an independent Kalman filter as described earlier. For example, in Figure 3.2, each of the four partials is tracked by an independent Kalman filter. Given the tracked amplitudes, the observed data at different regions are

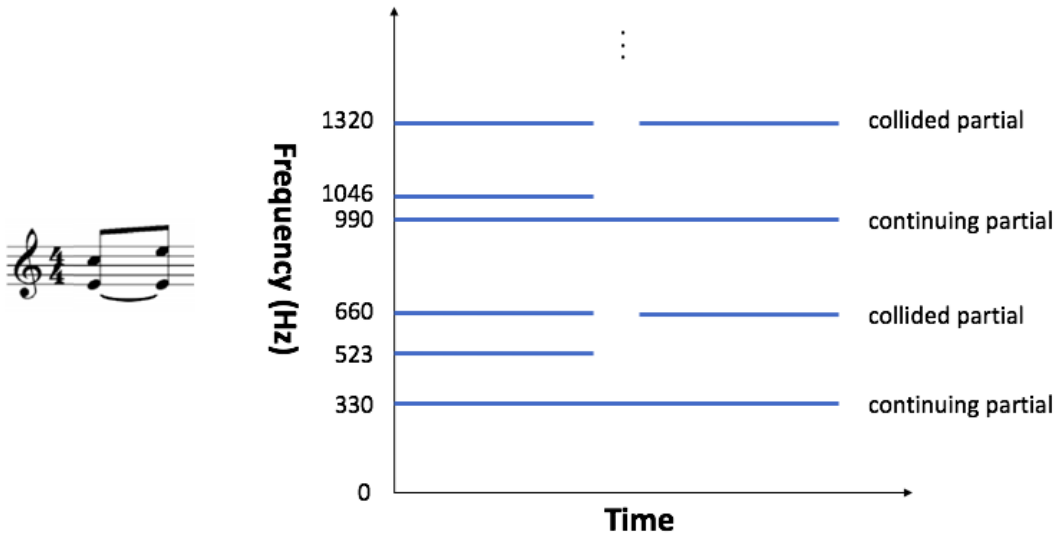


Figure 3.6: Two consecutive chords sharing a tied bottom note, E. The fundamental frequencies of the three notes are (from low to high): 330 Hz, 523 Hz, and 660 Hz.

independent:

$$p(\boldsymbol{y}_{n+1}|\boldsymbol{a}_{n+1}(x_{n+1})) = \prod_{p=1}^{P(x_{n+1})} p(\boldsymbol{y}^p_{n+1}|a^p_{n+1}(x_{n+1})) \tag{3.5}$$

Let's write the filtered probability distribution of the amplitudes at frame n as

$$a^p_n(x_n)|\boldsymbol{y}_{1:n} \sim N(m^p_n(x_n), v^p_n(x_n))$$

Then, according to Equations 3.1 and 3.2, we have

$$\boldsymbol{y}^p_{n+1}|a^p_{n+1}(x_{n+1}) \sim N(\boldsymbol{\mu}^p_{n+1}, \Sigma^p_{n+1})$$

where $\boldsymbol{\mu}^p_{n+1} = \lambda m^p_n(x_n)\boldsymbol{b}^p$ and $\Sigma^p_{n+1} = (\lambda^2 v^p_n(x_n) + \sigma^2)\boldsymbol{b}^p(\boldsymbol{b}^p)^T + \rho^2 I^p$, and I^p is an identity matrix with the same dimension as $\boldsymbol{b}^p(\boldsymbol{b}^p)^T$. That is to say:

$$p(\boldsymbol{y}^p_{n+1}|a^p_{n+1}(x_{n+1})) = \frac{1}{\sqrt{(2\pi)^{d^p}|\Sigma|}} exp(-\frac{1}{2}(\boldsymbol{y}^p_{n+1} - \boldsymbol{\mu}^p_{n+1})^T\Sigma^{-1}(\boldsymbol{y}^p_{n+1} - \boldsymbol{\mu}^p_{n+1})) \tag{3.6}$$

where d^p is the dimension of the vector $\boldsymbol{b}^p$. Therefore, we can calculate the data likelihood of $p(\boldsymbol{y}_{n+1}|\boldsymbol{a}_{n+1}(x_{n+1}))$ using Equations 3.5 and 3.6. In the case of transitioning to a new chord, $C(x_{n+1}) = C(x_n) + 1$, the calculation is very similar: for any continuing partial, it

has the same calculation as in Equation 3.6; for any newly formed partial, replace $\boldsymbol{\mu}_{n+1}^{p}$ with $m_0\boldsymbol{b}^p$, and Σ_{n+1}^{p} with $v_0 I^p$, the default distribution.

The hope is that, if the hypothesis state is true (if this state's corresponding chord is the actual chord sounding in the audio), after a few frames of tracking, the estimation of the amplitudes would fit the actual data quite well: the m_n^p's are close to the real data, and therefore, the resulting template is representative for the real data too. This way, the relative amplitudes among partials (the timbre information) in the template are "learned" from the data—a capability that doesn't exist in the fixed templates used in the HMM model.

3.4 Inference

At every frame, there are multiple discrete hypothesis states concerning the score position. A hypothesis state at frame n, x_n, is associated with a number of Gaussian distributions from the Kalman filters, each tracking one of chord $C(x_n)$'s partials. We can write the partial amplitudes with a state at frame n as $\boldsymbol{a}_n(x_n) = \{a_n^p(x_n)\}_{p=1}^{P(x_n)}$, but this notation is simplified as $\boldsymbol{a}_n$ in this section. Therefore, the joint filtered distribution of the state and the amplitudes can be represented as $p(x_n, \boldsymbol{a}_n|\boldsymbol{y}_{1:n})$. This is the distribution on the hidden variables at frame n after observing the data up to this frame, $\boldsymbol{y}_1, \ldots, \boldsymbol{y}_n$. This can be further written as

$$\begin{aligned} p(x_n, \boldsymbol{a}_n|\boldsymbol{y}_{1:n}) &= p(x_n|\boldsymbol{y}_{1:n})p(\boldsymbol{a}_n|x_n, \boldsymbol{y}_{1:n}) \\ &= p(x_n|\boldsymbol{y}_{1:n}) \prod_{p=1}^{P} p(a_n^p|x_n, \boldsymbol{y}_{1:n}) \end{aligned} \tag{3.7}$$

The second step in Equation 3.7 utilizes the independence assumption among partials, as discussed at the end of Section 3.3.1: the Kalman update for the partials depends on nonoverlapping spectrum regions in the data. In contrast to the HMM filtered distribution, the switching Kalman filtered distribution here is given by a discrete state probability, $p(x_n|\boldsymbol{y}_{1:n})$, *and* a product of Gaussian densities on the $\{a_n^p\}$, each of which is independent from the others given the state and the data.

The joint filtered distribution in Equation 3.7 evolves from frame n to $n+1$, and can be

calculated iteratively. According to the dependency graph in Figure 3.5, we have

$$\begin{aligned} p(x_{n+1}, x_n, \boldsymbol{a}_{n+1}, \boldsymbol{a}_n | \boldsymbol{y}_{1:n+1}) &= p(x_{n+1}, x_n, \boldsymbol{a}_{n+1}, \boldsymbol{a}_n, \boldsymbol{y}_{n+1} | \boldsymbol{y}_{1:n}) / p(\boldsymbol{y}_{n+1} | \boldsymbol{y}_{1:n}) \\ &\propto p(x_{n+1}, x_n, \boldsymbol{a}_{n+1}, \boldsymbol{a}_n, \boldsymbol{y}_{n+1} | \boldsymbol{y}_{1:n}) \\ &= p(x_n, \boldsymbol{a}_n | \boldsymbol{y}_{1:n}) p(x_{n+1} | x_n) p(\boldsymbol{a}_{n+1} | \boldsymbol{a}_n, x_n, x_{n+1}) p(\boldsymbol{y}_{n+1} | x_{n+1}, \boldsymbol{a}_{n+1}) \end{aligned}$$

where the first factor is the filtered joint distribution at frame n, and the other three factors are described in the last section. Getting the joint filtered distribution at frame $n+1$ includes two steps on $p(x_n, x_{n+1}, \boldsymbol{a}_n, \boldsymbol{a}_{n+1} | y_{1:n+1})$: marginalizing out the partial amplitudes, $\boldsymbol{a}_n$, and marginalizing out the state, x_n. These two steps are represented in Equation 3.8 and Equation 3.9 respectively.

Let's look at the amplitudes first. As mentioned in Section 3.3.2, the amplitudes evolve in two different fashions. In the case of a continuing chord where $C(x_{n+1}) = C(x_n)$, partials amplitudes in $\boldsymbol{a}_{n+1}$ and $\boldsymbol{a}_n$ share the same structure. Using the independence assumption on the partials, we have

$$p(x_{n+1}, x_n, \boldsymbol{a}_{n+1}, \boldsymbol{a}_n | y_{1:n+1}) = \prod_{p=1}^{P} p(x_{n+1}, x_n, a_{n+1}^p, a_n^p | y_{1:n+1})$$

Marginalizing out a_n^p leads to the usual Kalman filter update procedure described in Equations 3.3 and 3.4. Say $a_n^p(x_n) \sim N(m_n^p, v_n^p)$, and then the mean and variance of $a_{n+1}^p(x_{n+1})$ would be

$$\begin{aligned} m_{n+1}^p &= \lambda m_n^p + \frac{s_n^p}{\rho^2 + s_n^p}((\boldsymbol{b}^p)^T \boldsymbol{y}_n^p - \lambda m_n^p) \\ v_{n+1}^p &= s_n^p - \frac{(s_n^p)^2}{\rho^2 + s_n^p} \end{aligned} \tag{3.8}$$

where $s_n^p = \lambda^2 v_n^p + \sigma^2$. This way, we get $p(x_{n+1}, x_n, \boldsymbol{a}_{n+1})$ according to the usual update formula of the Kalman filter, applied independently to each partial; each Kalman filter calculates the distribution of a_{n+1}^p from the filtered distribution of a_n^p and the newly observed data, $\boldsymbol{y}_{n+1}$. In doing so, the distribution for each partial a_{n+1}^p is conditioned on $\boldsymbol{y}_{n+1}^p$—the

relevant (and non-overlapping) portion of $\boldsymbol{y}_{n+1}$. In the other case, if it is a new chord where $C(x_{n+1}) = C(x_n) + 1$, the amplitudes of new partials adopt the default distribution $N(m_0, v_0)$ regardless of the distribution of $\boldsymbol{a}_n$, and the continuing partials follow the same update procedure as in the case of a continuing chord. Therefore, in the latter case, too, we can compute the value of $p(x_{n+1}, x_n, \boldsymbol{a}_{n+1})$ in a straightforward manner.

The above discussion shows how to marginalize out the continuous variables $\boldsymbol{a}_n$ through the standard Kalman filter formulation. The difficulty of implementing a switching Kalman filter arises when further marginalizing out the discrete variable, x_n, by

$$p(x_{n+1}, \boldsymbol{a}_{n+1} | \boldsymbol{y}_{1:n+1}) = \sum_{x_n : \mathcal{P}(x_{n+1})} p(x_{n+1}, x_n, \boldsymbol{a}_{n+1} | \boldsymbol{y}_{1:n+1}) \tag{3.9}$$

where $\mathcal{P}(x_{n+1})$ is the set of predecessors of state x_{n+1} in the state graph. The difficulty is that different predecessors in $\mathcal{P}(x_{n+1})$ are associated with different estimates of the amplitudes represented by $\boldsymbol{a}_{n+1}$. When summing out all possible predecessors as in Equation 3.9, the estimate of each partial's amplitude, a^p_{n+1}, becomes a Gaussian mixture distribution. The number of components grows exponentially with n, making the problem intractable without approximation. We can use the approach familiar in the switching Kalman filter literature, and approximate the mixture of multiple Gaussians by a single Gaussian with the same mean and variance as in the mixture (see the *moment matching* method in Section 2.6.2). Figure 3.7 demonstrates this process.

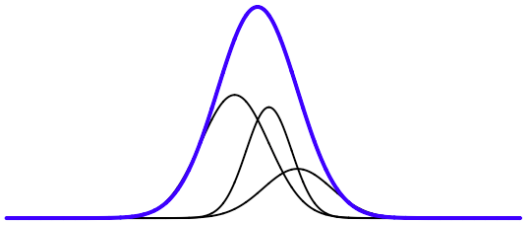


Figure 3.7: An illustration for approximating a mixture of three Gaussians by a single Gaussian (drawn in thicker blue line).

Since the state graph is a chain of micro states with self-loops, the growth of the discrete states can be represented as a binary tree, as in the left tree in Figure 3.8: a left child

represents looping back to the state, and a right child represents moving on to the next state. Marginalizing out the discrete states in every frame (a level in the tree) means consolidating the information about a state at a frame regardless of its ancestors in previous frames. For example, at the fourth level of the right "tree" (directed graph) in Figure 3.8, the state with index of "3" has three state paths that lead to it, and each path might result in different estimations of the partial amplitudes—the deeper the tree is, the more paths there are leading to a node. Using a single Gaussian to approximate and summarise these estimations guarantees that a node in the (right) tree always has one Gaussian estimation for a partial, no matter how deep this node is. However, the tree still grows linearly with time. I use the *beam search* technique to address this by limiting the number of hypotheses to be constant: at every level, only keep no more than a fixed number of nodes that have the highest filtered discrete probabilities, and prune out the rest.

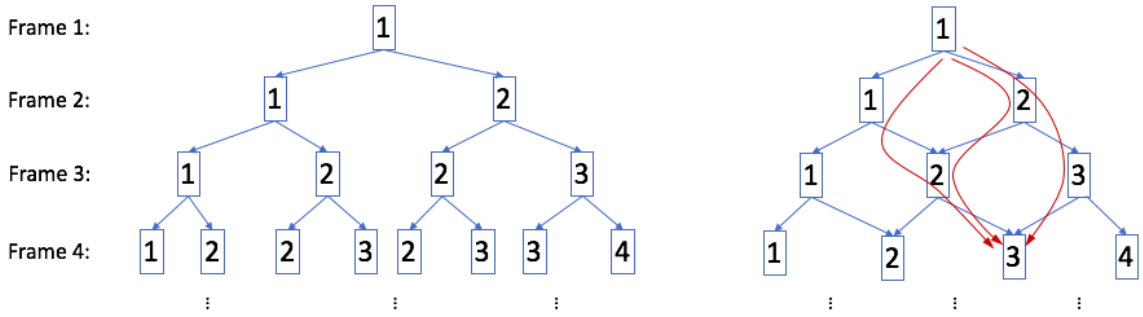
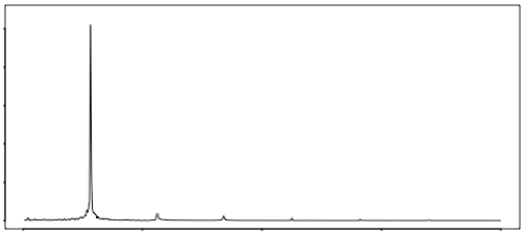
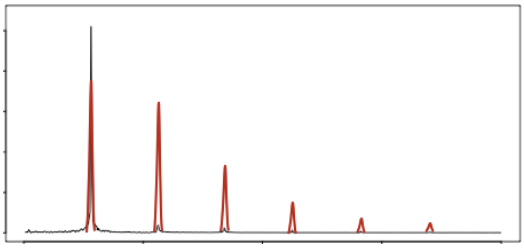


Figure 3.8: Computational graph.

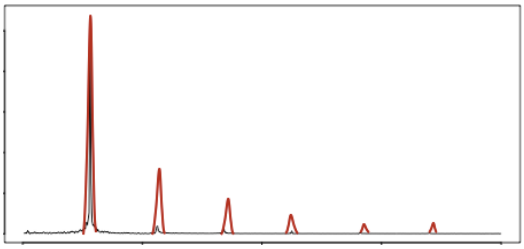
Allowing templates to adapt to the observed data could also be a double-edged sword. At any given frame, there are multiple discrete hypotheses, including both correct and incorrect ones. The templates for both kinds of hypotheses are "improved" to match the observed data better. For a correct hypothesis state that corresponds with the actual chord sounding in the audio, the template improvement is illustrated in Figure 3.9, where the mean of the estimated amplitudes are drawn. The spectrum in this frame indicates that this note's first partial has much more energy than its higher partials; the template after observing the data shows relatively lower energy for higher partials than the original template, because the



(a) Spectrum of a single note observed at an audio frame (the data).



(b) Template (red) *before* observing data.



(c) Improved template (red) *after* observing data.

Figure 3.9: Illustration of template adaptation for a correct hypothesis state.

template has adapted to the observed data. On the other hand, for an incorrect hypothesis state, the partial structure of its template is usually different from the real sounding chord; therefore, it oftentimes has to ignore some "peaks" in the observed data simply because its structure doesn't permit it to have any energy in those frequency regions. The idea is that, on the whole, the correct hypothesis states would adapt to the data *better* than the incorrect ones, resulting in a data model that can discriminate those two kinds more easily. I reconsider this optimism in the context of the experiments, and discuss it in more detail in Chapter 5.

CHAPTER 4

Modeling Tempo

4.1 Linear Dynamical System

The polyphonic score is represented as a sequence of chords, with a new chord appearing whenever any note is added, ended, or changed in the current chord. Let's assume there are K such chords in the musical score, each chord with a nominal musical length (e.g., 1/4 for a quarter note and 1 for a whole note) represented by l_k, $k = 1, \ldots, K$. The chord index, k, acts as the discretized "time step" for our linear dynamical system described in this section.

I assume every chord has its own tempo value, and that value doesn't change within a chord's lifetime. Let t_k be the tempo of the kth chord (seconds per whole note), and o_k be this chord's onset time (in seconds), where $k = 1, \ldots, K$. The joint evolution of the tempo and the onset is modeled as a linear dynamical system. If we treat the onsets as observable data (they are not, and the real observable data are incorporated later in Section 4.2), the formula is the same as a Kalman filter model:

$$o_{k+1} = o_k + l_k t_k + \varepsilon_{k+1} \tag{4.1}$$

$$t_{k+1} = t_k + \eta_{k+1} \tag{4.2}$$

where all random variables have normal distributions:

$$\begin{aligned}
o_1 &\sim N(\mu_{o,1}, \sigma^2_{o,1}) \\
t_1 &\sim N(\mu_{t,1}, \sigma^2_{t,1}) \\
\varepsilon_k &\sim N(0, \sigma^2_{\varepsilon,k}), \ k = 2, \ldots, K \\
\eta_k &\sim N(0, \sigma^2_{\eta,k}), \ k = 2, \ldots, K
\end{aligned}$$

The ε_k's and η_k's are all mutually independent, and they are independent from o_1 and t_1 as

well. The dependency graph is in Figure 4.1.

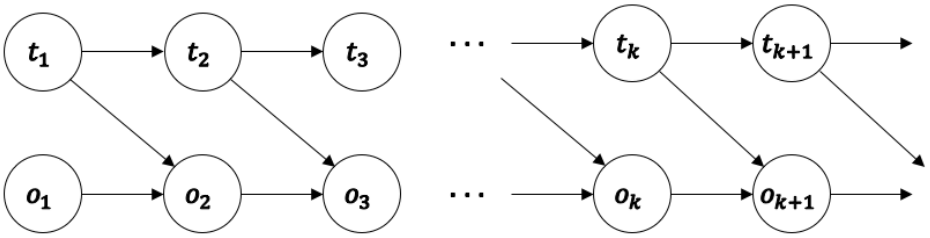


Figure 4.1: Linear dynamical system of the tempo and the onset.

For the rest of this chapter, I refer to a "chord" (in our polyphonic score representation) as a "note" for simplicity's sake.

4.1.1 Marginal Likelihood of Onsets

This linear dynamical system can be viewed as a Markov process of generating the onsets, $o_1^K = (o_1, o_2, \ldots, o_K)$, demonstrated as follows. According to the chain rule and the described model, we can write

$$\begin{aligned} p(o_1^K) &= p(o_1) \prod_{k=1}^{K-1} p(o_{k+1}|o_1^k) \\ &= p(o_1) \prod_{k=1}^{K-1} \int_{t_k} p(o_{k+1}|t_k, o_k) p(t_k|o_1^k) \, \mathrm{d}t_k \end{aligned}$$

Because $o_{k+1} = o_k + l_k t_k + \varepsilon_{k+1}$, the integral factors can be further simplified as

$$\begin{aligned} \int_{t_k} p(o_{k+1}|t_k, o_k) p(t_k|o_1^k) \, \mathrm{d}t_k &= \int_{t_k} N(o_{k+1}; o_k + l_k t_k, \sigma^2_{\varepsilon,k+1}) N(t_k; \mu(o_1^k), \sigma^2(o_1^k)) \, \mathrm{d}t_k \\ &= N(o_{k+1}; o_k + l_k \mu(o_1^k), \sigma^2_{\varepsilon,k+1} + l_k^2 \sigma^2(o_1^k)) \end{aligned}$$

where $\mu(o_1^k) = E(t_k|o_1^k)$ and $\sigma^2(o_1^k) = Var(t_k|o_1^k)$, which can be iteratively calculated using a Kalman filter (see Section 2.5). Thus, we have

$$p(o_1^K) = p(o_1) \prod_{k=1}^{K-1} N(o_{k+1}; o_k + l_k \mu(o_1^k), \sigma^2_{\varepsilon,k+1} + l_k^2 \sigma^2(o_1^k)) \tag{4.3}$$

which can be iteratively calculated as k increases from 1 to $K-1$.

4.2 Frame-wise Representation

The previous discussions are based on the linear dynamical system at the note level, as depicted in Figure 4.1, where the discretized "time step" is the note index. However, in real-world applications, the audio is received frame by frame (with a hop size of 16 milliseconds). In order to incorporate such audio frames as observed data, we have to change the scope and view the model in Figure 4.1 to the *frame* level, with each audio frame as a time step. Let n denote the index of a frame, $n = 1, \ldots, N$, where N is the total number of frames in the audio. Any frame n has a label variable, $k_n \in \{0, \ldots, K\}$, which is the index of the sounding note at that frame. Frames before the first note being played are labeled as 0, i.e., $\{n : k_n = 0\}$. Denote y_n as the observed audio at the nth frame (unlike in the baseline and the timbre tracking methods, here I use an unbolded symbol to represent the data). We can assume the distribution of the audio frame data only depends on this frame's label—which note is being played. Figure 4.2 shows the model at both levels in the same graph.

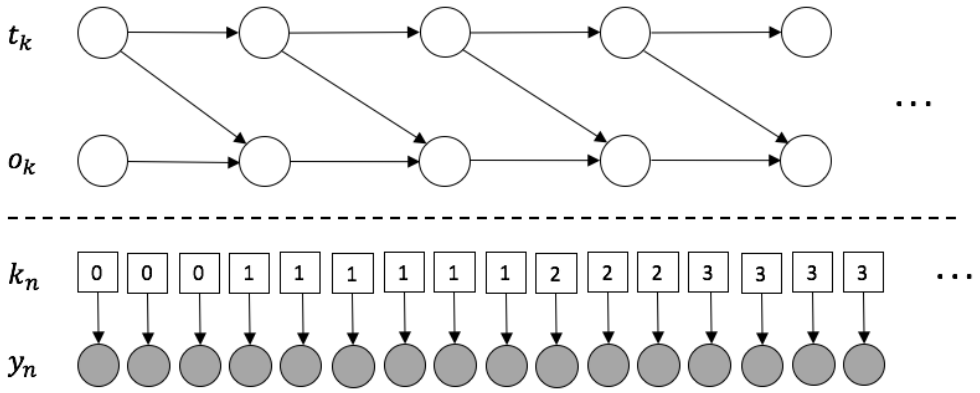


Figure 4.2: Dependency graph among the variables (modified from Raphael (2004)). The upper panel is at the note level; the lower panel is at the frame level. The circles are continuous variables and the squares are discrete variables. The observed variables are shaded.

The note onsets and the frame labels are nearly interchangeable. Let Δ be the time difference between adjacent frames in milliseconds (i.e., the hop size); a sequence of frame

labels, k_1^n, can then be recovered from a sequence of onsets, o_1^k, or vice versa, like this:

$$k_n = \min\left\{k \in \{0, \ldots, K\} : \ n\Delta < o_{k+1}\right\} \tag{4.4}$$

$$o_k \approx \Delta \cdot \min\left\{n \in \{1, \ldots, N\} : \ k_n = k\right\} \tag{4.5}$$

The note-wise representation and the frame-wise representation are almost equivalent, except that there are two additional assumptions in the latter:

> **Assumption 1**. The note onsets are now discrete because they have to be multiples of Δ, as in Equation 4.5.
>
> **Assumption 2**. A note must last at least one frame, so $o_{k+1} - o_k \geq \Delta$.

Because the labels and the onsets can be derived deterministically from each other as above, the onsets can actually be viewed as "discrete" variables, and the tempo variables are the only real continuous (hidden) variables.

4.2.1 Marginal Likelihood of Labels

The Markov process of generating the frame labels, k_1^N, is (almost) equivalent to generating the note onsets in Section 4.1.1, but at the frame level. Using the chain rule, we have

$$p(k_1^N) = p(k_1) \prod_{n=1}^{N-1} p(k_{n+1}|k_1^n) \tag{4.6}$$

Following the assumptions in the frame-wise representation, there could be only two possible values for k_{n+1} in each factor $p(k_{n+1}|k_1^n)$: either k_n if it "decides" to stay at the current note, or $k_n + 1$ if it "decides" to move on to the next note.

From Equation 4.3, we know that the onset of the pending note $k_n + 1$, given all previous onsets $o_1^{k_n}$ (equivalent to k_1^n), has a density function

$$p(o_{k_n+1}|o_1^{k_n}) = N(o_{k_n+1}; o_{k_n} + l_{k_n}\mu(o_1^{k_n}), \sigma^2_{\varepsilon,k_n+1} + l^2_{k_n}\sigma^2(o_1^{k_n}))$$

Writing ϕ as the standard normal density function, the above density is expressed by

$f(o_{k_n+1}; o_{k_n} + l_{k_n}\mu(o_1^{k_n}), \sigma^2_{\varepsilon,k_n+1} + l^2_{k_n}\sigma^2(o_1^{k_n}))$, where $f(\cdot)$ is defined as:

$$f(x; \mu, \sigma) = \phi(\frac{x-\mu}{\sigma})$$

We can use this density function to compute $p(k_{n+1}|k_1^n)$ which has two cases:

$$p(k_{n+1}|k_1^n) = \begin{cases} p_1, & k_{n+1} = k_n \ (same\ note) \\ 1-p_1, & k_{n+1} = k_n + 1 \ (new\ note) \end{cases} \tag{4.7}$$

We can focus on calculating the first case, while the second case has the complimentary probability of p_1. In the first case, where the process stays in the same note at frame $n+1$, we know that the onset of the next note will be *after* frame $n+1$, given we already know that the onset must be after frame n. Therefore, p_1 is a conditional probability:

$$p_1 = P(o_{k_n+1} > (n+1)\Delta \mid o_{k_n+1} > n\Delta\ , o_1^{k_n}) \tag{4.8}$$

Writing Φ as the cumulative distribution function of ϕ, we have

$$p_1 = \frac{1 - \Phi(\frac{c+\Delta-\mu}{\sigma})}{1 - \Phi(\frac{c-\mu}{\sigma})} \tag{4.9}$$

where $\mu = o_{k_n} + l_{k_n}\mu(o_1^{k_n})$, $\sigma = \sqrt{\sigma^2_{\varepsilon,k_n+1} + l^2_{k_n}\sigma^2(o_1^{k_n})}$, and $c = (n+1)\Delta$.

Therefore, we can calculate the probability of any label sequence $p(k_1^N)$ iteratively as in Equation 4.6, by using Equation 4.7. Up to this point, we have two nearly equivalent generative models: the one in Section 4.1.1 computes the probability of an onset sequence iteratively at the note level, while the one here computes the probability of a label sequence iteratively at the frame level.

4.2.2 Note Duration and Note Age

In Section 4.2.1, Equation 4.7 is calculated based on the distribution of the onset for the pending note, $k_n + 1$. In this section, I introduce two variables—note duration and note age—and calculate Equation 4.7 from a slightly different perspective.

The duration of the kth note is the difference between its two adjacent onsets:

$$L_k = o_{k+1} - o_k = l_k t_k + \varepsilon_{k+1}$$

As mentioned before in Section 4.1.1, given the onsets o_1^k, the tempo of the last note in the sequence has a Gaussian distribution, and we can use a Kalman filter to calculate its mean and variance, $\mu(o_1^k) = E(t_k|o_1^k)$ and $\sigma^2(o_1^k) = Var(t_k|o_1^k)$. Thus, L_k also has a Gaussian distribution with its mean as $l_k\mu(o_1^k)$ and its variance as $\sigma^2_{\varepsilon,k+1} + l_k^2\sigma^2(o_1^k)$, given o_1^k. In the frame-wise representation, a note's duration, L_{k_n}, has the additional requirements that it should be multiples of Δ, and be at least Δ (milliseconds) long.

We define a note's age as the number of frames this note has been through so far as of the nth frame, denoted as a_n. The age variable, capable of representing partial notes, can only exist here, in the frame-wise representation, because the note-wise representation (Figure 4.1) operates at the scale of complete notes. The age of the note k_n can be calculated by

$$a_n = n - \frac{o_{k_n}}{\Delta} + 1$$

To calculate p_1 in Equation 4.7, we can rewite Equation 4.9 from the perspective of the note length: given that this note has lasted a_n frames, what's the probability of it lasting for at least $a_n + 1$ frames? It can be expressed as

$$p_1 = P(L_{k_n} \geq (a_n + 1)\Delta \mid L_{k_n} \geq a_n\Delta\,, o_1^{k_n}) = \frac{1 - \Phi(\frac{(a_n+1)\Delta-\mu}{\sigma})}{1 - \Phi(\frac{a_n\Delta-\mu}{\sigma})} \tag{4.10}$$

where $\mu = l_{k_n}\mu(o_1^{k_n})$ and $\sigma = \sqrt{\sigma^2_{\varepsilon,k_n+1} + l^2_{k_n}\sigma^2(o_1^{k_n})}$. This is consistent with the result we get in Section 4.2.1.

4.3 Computation

The above section describes the model at the frame level. The number of possible label sequences, however, grows exponentially with n, $O(2^n)$, as illustrated by the binary tree structure in Figure 4.3. In this section, we discuss the computational aspects of the model:

how to determine the filtered probability of a label, given all observed audio data up to the current frame, and how to reasonably approximate the calculation so it becomes tractable.

4.3.1 Tree Representation

The tree in Figure 4.3 represents all possible label sequences, $\{k_1^N\}$. At any frame n, each node not only has a label indicating its note index, k_n, it also includes the age information of this note—the number of frames this note has been through so far—denoted by a_n. From a label sequence k_1^n, we can determine the corresponding age sequence a_1^n, and vice versa. We include the age variable in the tree because we need it in the generative model, as indicated in Equation 4.10. A node has two children: a left child means it stays in the same note and thus the age increases by one frame, and a right child means it moves on to the next note and thus the age resets to be 1. A node in the tree actually represents a label sequence with the path that leads to this node from the root. For example, the red dotted line in Figure 4.3 represents the label sequence of $k_1^6 = 011122$ (or the age sequence of $a_1^6 = 112312$). A path can be viewed as a sequence of decisions of choosing the left or right branch starting from the root node at frame 1 leading to the end node in the path.

For every node in the tree, denoted by its path of k_1^n, we can calculate two probabilities: the discrete probability of arriving at this node, $p(k_1^n)$, and the continuous probability distribution of the tempo at this node, $p(t_{k_n}|k_1^n)$. As discussed before, the tempo has a Gaussian distribution (drawn besides the nodes in the first four frames in Figure 4.3), given the path leading to this node. A Kalman filter keeps track of the tempo down the path, and updates its distribution when and only when the path chooses a right branch (drawn with thicker lines), i.e., whenever it observes a note onset (discussed in Section 4.1).

The probability of arriving at a node, $p(k_1^n)$, according to the frame-wise generative model, can be iteratively calculated by $p(k_1^n) = p(k_1^{n-1})p(k_n|k_1^{n-1})$. It means $p(k_1^n)$ can be calculated from two parts: the probability of arriving at its parent node at frame $n-1$, $p(k_1^{n-1})$, and the probability of choosing the left or the right branch when transitioning from

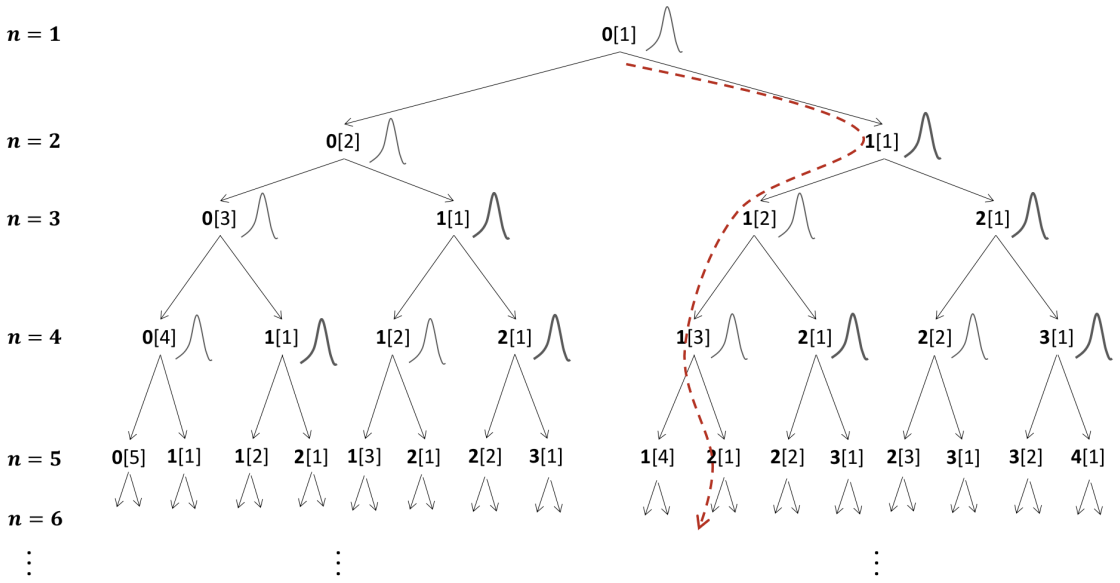


Figure 4.3: Exponential growth of label sequences. Each node in the tree has three aspects: the note index label, the age of this note (in the square brackets), and the distribution of the tempo. A node's left child always has the same label with its parent, and it is one frame older than its parent; a node's right child always has the label of the next note of its parent's note, and it always has the age of 1. From the root to any node in the tree forms a particular label sequence, e.g., from the root to the left child of 2[1] at frame 5 (the dotted line) forms the sequence of 011122... The tempo distributions at *onset* nodes are drawn with thicker lines.

frame $n-1$ to frame n, $p(k_n|k_1^{n-1})$. We can use either Equation 4.9 or Equation 4.10 to calculate the latter, and both equations require the distribution of the tempo at the parent node, $p(t_{k_{n-1}}|k_1^{n-1})$—or equivalently, $p(t_{k_{n-1}}|o_1^{k_{n-1}})$.

Adopting the iterative nature of the calculation, we can compute the probability of (arriving at) every node and its tempo distribution, frame by frame, starting from the root. Since the tree includes every possible label sequence, those probabilities give us $p(k_1^n)$ for all $\{k_1^n\}$, $n = 1, \ldots, N$.

4.3.2 Conditioning on Data

This section continues to focus on calculating the probability of arriving at a node in the tree, but now *conditioned* on the observed audio data up to the current frame—$p(k_1^n|y_1^n)$. Considering y_1^n ensures that sequences more consistent with the observed data will receive higher probabilities than the rest, thus helping us identify more likely sequences. On the other hand, the tempo will not be affected by the observed data since the tempo distribution at a node depends only on the corresponding label sequence, i.e., $p(t_{k_n}|k_1^n, y_1^n) = p(t_{k_n}|k_1^n)$.

The audio frame data $y_1, \ldots, y_n$ are assumed to be conditionally independent from each other given the frame labels $k_1, \ldots, k_n$:

$$p(y_1^n|k_1^n) = \prod_{i=1}^{n} p(y_i|k_i)$$

Therefore, we can have

$$\begin{aligned} p(k_1^n|y_1^n) &= \frac{1}{Z_n} p(k_1^n, y_1^n) \\ &= \frac{1}{Z_n} p(k_1^n) p(y_1^n|k_1^n) \\ &= \frac{1}{Z_n} p(k_1^n) \prod_{i=1}^{n} p(y_i|k_i) \end{aligned}$$

where

$$Z_n = p(y_1^n) = \sum_{\{k_1^n\}} p(k_1^n, y_1^n)$$

The joint probability $p(k_1^n, y_1^n)$ can be calculated iteratively from $p(k_1^{n-1}, y_1^{n-1})$ by

$$p(k_1^n, y_1^n) = p(k_1^{n-1}, y_1^{n-1}) p(k_n | k_1^{n-1}) p(y_n | k_n)$$

The calculation of the middle factor $p(k_n | k_1^{n-1})$ is discussed in Equations 4.7, 4.9, and 4.10, which involve using the tempo distribution of $p(t_{k_{n-1}} | k_1^{n-1})$. The third factor $p(y_n | k_n)$ is the data likelihood of the nth frame, discussed in Section 2.4.2. Therefore, we can calculate $p(k_1^n, y_1^n)$ for every node in the tree iteratively from frame 1 to frame N. To get the filtered probability of $p(k_1^n | y_1^n)$, we simply normalize $p(k_1^n, y_1^n)$ with Z_n at every frame. In sum, with the help of the data model factors $\prod_{i=1}^{n} p(y_i | k_i)$, we should be able to distinguish the sequences better by using $p(k_1^n | y_1^n)$ instead of $p(k_1^n)$: hypothesized sequences that are closer to the true sequence should have larger values of $p(k_1^n | y_1^n)$ than sequences further from the truth.

4.3.3 Filtering

The task of filtering is to compute $p(k_n, t_{k_n} | y_1^n)$: the joint distribution of the last hidden states in the sequence, given the sequence of observed data so far. Writing $\mathcal{K}(k_n)$ as the set of all label sequences in Figure 4.3 that are n labels long and end with k_n, this filtered probability is:

$$p(k_n, t_{k_n} | y_1^n) = \sum_{k_1^n \in \mathcal{K}(k_n)} p(k_1^n, t_{k_n} | y_1^n) \tag{4.11}$$

$$= \sum_{k_1^n \in \mathcal{K}(k_n)} p(k_1^n | y_1^n) p(t_{k_n} | k_1^n) \tag{4.12}$$

This is a Gaussian mixture distribution with $|\mathcal{K}(k_n)|$ components. Without approximation, the computation is intractable because the size of $\mathcal{K}(k_n)$ grows exponentially with n. In this section, I discuss how to make these calculations tractable, first by borrowing ideas from the filtering process in the hidden Markov model.

In Section 2.4, the state sequences in the HMM can be represented by a tree too, as in

Figure 4.4. In the left tree, the number of state sequences grows exponentially. Because of the Markov property, nodes sharing the same state label at a time frame proceed exactly the same in the future, with the same calculations: they have the same children (and subtrees) with the same transition probabilities. To reduce this redundancy, we can "merge" those nodes at every frame, and keep only one copy of a state at any frame. This operation results in the "tree" (directed graph) on the right, which grows only linearly. A merged node has the summed probabilities from its two merging nodes.

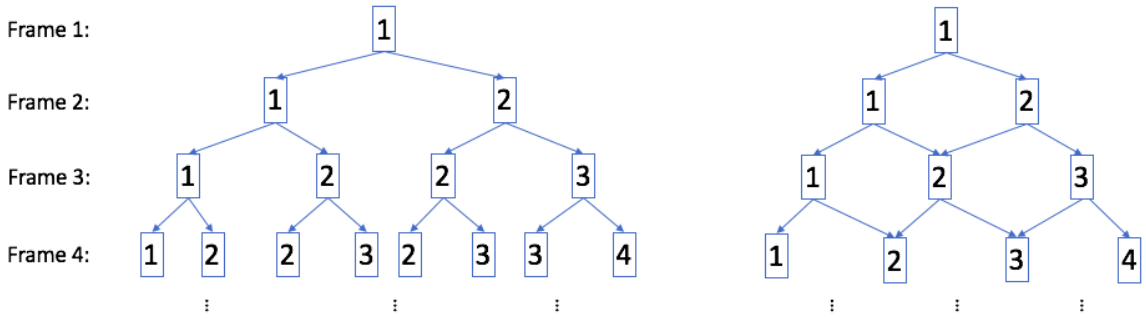


Figure 4.4: Merging in the HMM. The numbers in the tree nodes are the indices of the states in a Markov chain. The number of nodes grows exponentially with time in the left tree, while the right tree grows linearly because of merging.

However, there are two caveats if we copy the same merging operation to the frame-wise model in Figure 4.3. First, at a given frame, there can be nodes sharing the same label state but with different ages, and these nodes do *not* proceed the same because the transition probabilities depend on the age variable (as shown in Equation 4.10). Therefore, we cannot just merge nodes with the same label like in the HMM. Instead, we can only merge nodes sharing both a state label and an age, which results in quadratic growth of the tree (although still an improvement over exponential growth). The merged node has the summed probability:

$$p(\text{merged node}) = \sum_{\text{nodes}} p(\text{node}_i)$$

Second, and more problematic, each node in the frame-wise model also carries a Gaussian distribution for the tempo. When merging two or more nodes together at a frame,

the merged node would then carry a Gaussian mixture distribution:

$$p(t_{\text{merged}}) = \sum_{\text{nodes}} \frac{p(\text{node}_i)}{p(\text{merged node})} p(t \mid \text{node}_i)$$

As more and more merging operations happen over time, the number of components in a tempo distribution grows exponentially, with each component corresponding with a possible label sequence, and the problem remains intractable. We can solve this problem by approximating a Gaussian mixture with a single Gaussian that has the same mean and variance as the Gaussian mixture (discussed in Section 2.6.2). This approximation is reasonable if the components with larger weights are close to each other in terms of mean and variance, and if those further away have smaller weights, making their impact negligible, as illustrated in the left panel of Figure 4.5. It's reasonable to believe that reality more often reflects the case as seen in the left panel than the right one, because unlikely nodes tend to have smaller probabilities and thus smaller weights in the mixture, and can be ignored safely, while more likely nodes tend to have more similar estimates of the tempo because they lean towards the truth. The next section describes the algorithm for calculating the filtered probabilities with approximation.

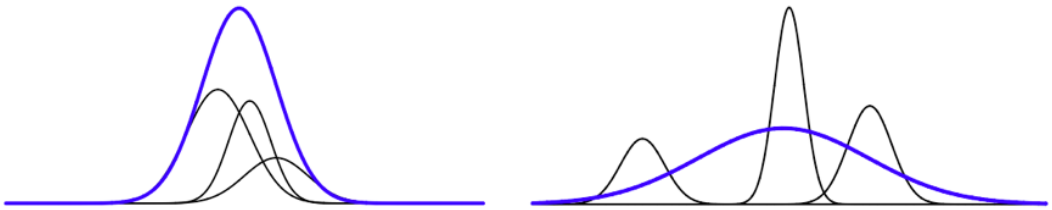


Figure 4.5: Two different scenarios when using a single Gaussian (with a thicker line) to approximate a Gaussian mixture. The left panel is more desirable than the right panel because an approximation is more reasonable when the components are more similar to each other (or the distribution is more skewed towards components with larger weights).

4.3.4 Filtering with Approximation

As discussed above, to limit the exponential growth of the tree, we merge multiple nodes that share the same label and age, (k_n, a_n), into one node at each frame, and we approximate the

resulting Gaussian mixture distribution of the tempo with a single Gaussian. The process is demonstrated in Figure 4.6 and Figure 4.7.

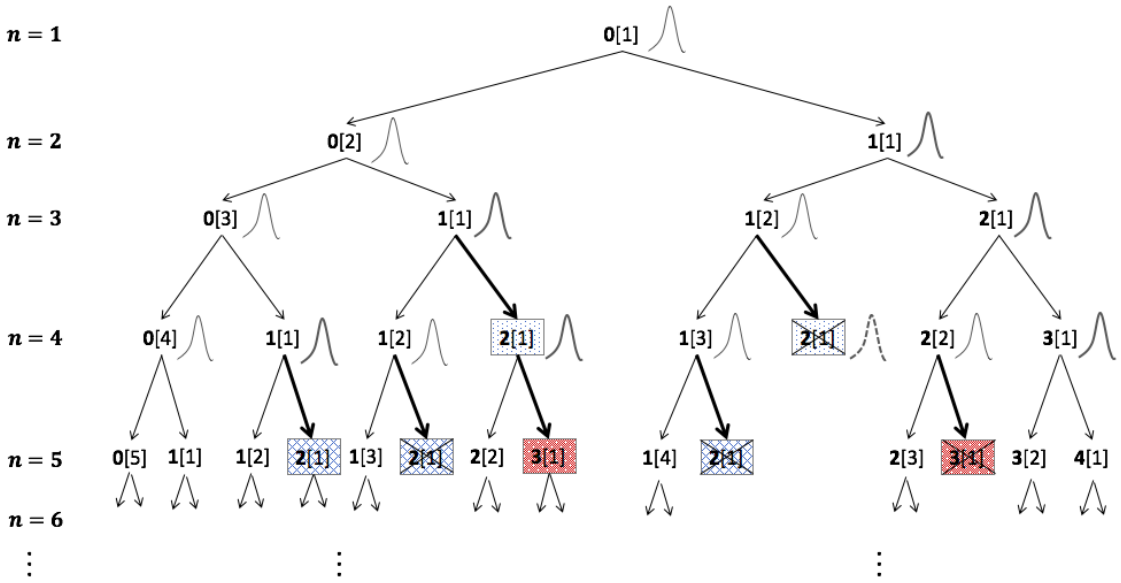


Figure 4.6: Limiting the tree's growth by merging nodes with the same label and age. In this graph, a group of merging nodes are shaded with the same pattern; only one of them continues to split in the next frame, and the rest of the duplicates are crossed out. Duplicated nodes are always the right children with the age of 1, emphasized by the thickened arrows.

Under this strategy, a left child is guaranteed to have no duplicates within a frame. All left children's ages are at least two (frames), because they represent continuing notes. By the same logic, a node with its age larger than one must be a left child. When proceeding from frame $n-1$ to n, suppose there are two nodes at frame n both identified as $k_n[a_n]$ with $a_n \geq 2$; then there must be two parent nodes identified as $k_n[a_n - 1]$ at frame $n-1$, contradicting the rule that there is only one copy of the node with a certain label and age at a frame. Therefore, left children do not need to be merged because they are always the only copy at a frame.

Merging nodes are always the right children of their parent nodes, and these nodes always have the age of 1. For nodes identified as $k_n[1]$ at frame n, their parent nodes must have the label of $k_n - 1$ and could have the age of $1, 2, \ldots, n - k_n$ from frame $n-1$. Therefore, there are $n - k_n$ copies of nodes $k_n[1]$ being merged together at frame n. It's

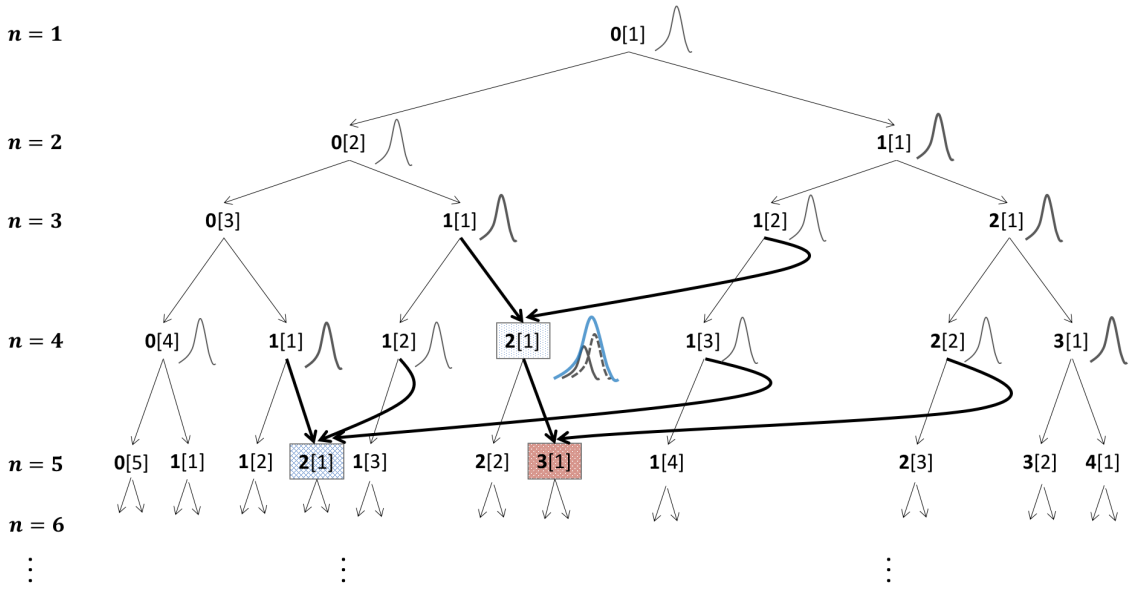


Figure 4.7: Three merging operations happening in the first five frames. The first merge happens at frame #4, when the right child of 1[1] and the right child of 1[2] (from frame #3 to frame #4) merges into one node. This merged node has a Gaussian mixture distribution for the tempo with two components from the two copies of the 2[1] (at frame #4 in Figure 4.6), which is approximated by a single Gaussian (drawn with a thicker blue line). The two merges at frame #5 have similar approximations.

worth noting that the size of the merging group, $n - k_n$, could be 1, which means there is no merging happening.

Distinguishing those two cases, we can iteratively approximate the discrete filtered probability of the label and the age, $p(k_n, a_n|y_1^n)$, and the continuous filtered probability of the tempo, $p(t_{k_n}|k_n, a_n, y_1^n)$. From previous discussions, we know that $p(t_{k_n}|k_n, a_n, y_1^n)$ is always a single Gaussian (after approximation), and we write its mean and variance as $\mu_n = \mu(k_n, a_n, y_1^n)$ and $\sigma_n^2 = \sigma^2(k_n, a_n, y_1^n)$.

The discrete filtered probability can be iteratively calculated as in the following equation, in which the second case represents the merging operation by summing the probabilities of

the duplicated nodes at frame n. To wit,

$$
\begin{aligned}
p(k_n, a_n|y_1^n) =& \frac{1}{W_n} p(k_n, a_n, y_n|y_1^{n-1}) \\
=& \begin{cases} \frac{1}{W_n} p(k_{n-1}, a_{n-1}|y_1^{n-1})\ p(k_n, a_n|k_{n-1}, a_{n-1}, y_1^{n-1})\ p(y_n|k_n)\ , \\ \qquad\qquad\qquad\qquad\qquad\qquad k_n = k_{n-1}\ and\ a_n > 1\ (same\ note) \\ \frac{1}{W_n} \sum\limits_{\substack{1 \le a_{n-1} \\ \le n - k_n}} p(k_{n-1}, a_{n-1}|y_1^{n-1})\ p(k_n, a_n|k_{n-1}, a_{n-1}, y_1^{n-1})\ p(y_n|k_n)\ , \\ \qquad\qquad\qquad\qquad\qquad\qquad k_n = k_{n-1} + 1\ and\ a_n = 1\ (new\ note) \end{cases}
\end{aligned}
\tag{4.13}
$$

where

$$
W_n = p(y_n|y_1^{n-1}) = \sum_{\{k_n, a_n\}} p(k_n, a_n, y_n|y_1^{n-1})\ ,
$$

and the middle factor

$$
p(k_n, a_n|k_{n-1}, a_{n-1}, y_1^{n-1}) = \begin{cases} p(o_{k_{n-1}+1} > n\Delta | o_{k_{n-1}+1} > (n-1)\Delta)\ , \\ \quad k_n = k_{n-1}\ and\ a_n > 1\ (same\ note) \\ p(o_{k_{n-1}+1} \le n\Delta | o_{k_{n-1}+1} > (n-1)\Delta)\ , \\ k_n = k_{n-1} + 1\ and\ a_n = 1\ (new\ note) \end{cases} \tag{4.14}
$$

where $o_{k_{n-1}+1}$ has a Gaussian distribution, with its mean as $(n - a_{n-1})\Delta + l_{k_{n-1}}\mu_n$, and its variance as $l_{k_{n-1}}^2\sigma_n^2 + \sigma_{\varepsilon, k_{n-1}+1}^2$.

The filtered tempo can also be calculated iteratively, according to two cases. In the case where it stays in the same note, the filtered tempo doesn't change, i.e.,

$$
p(t_{k_n}|k_n, a_n, y_1^n) = p(t_{k_{n-1}}|k_{n-1}, a_{n-1}, y_1^{n-1}) \tag{4.15}
$$

when $k_n = k_{n-1}$ and $a_n = a_{n-1} + 1$, and we simply retain the computed mean and variance. In the other case where it moves on to the next note, so $k_n = k_{n-1} + 1$ and $a_n = 1$, the filtered tempo becomes a Gaussian mixture:

$$
\begin{aligned}
& p(t_{k_n}|k_n, a_n, y_1^n) \\
& \quad = \frac{1}{W_n} \sum_{\substack{1 \le a_{n-1} \\ \le n - k_n}} p(k_{n-1}, a_{n-1}|y_1^{n-1}) \cdot p(k_n, a_n|k_{n-1}, a_{n-1}, y_1^{n-1}) \cdot \\
& \qquad\qquad p(y_n|k_n) \cdot p(t_{k_n}|o_{k_n} = n, o_{k_{n-1}} = n - a_{n-1}, \mu_{n-1}, \sigma_{n-1}^2)
\end{aligned}
\tag{4.16}
$$

where the last factor can be calculated by the the Kalman filter discussed in Section 4.1.1. We further approximate the Gaussian mixture $p(t_{k_n}|k_n, a_n, y_1^n)$ using a single Gaussian with the mean as μ_n and the variance as σ_n^2 (the approximation technique discussed in Section 2.6.2).

Putting Equations 4.13, 4.15, and 4.16 together, we can iteratively calculate the joint filtered probability $p(k_n, a_n, t_{k_n}|y_1^n)$ by

$$p(k_n, a_n, t_{k_n}|y_1^n) = p(k_n, a_n|y_1^n)\ p(t_{k_n}|k_n, a_n, y_1^n) \tag{4.17}$$

Even with merging, however, the number of hypotheses still grows quadratically as n increases—the first caveat noted in Section 4.3.3. To make the computation feasible, I use the beam search technique, and keep, at any frame, no more than a fixed number of hypotheses with the highest filtered probabilities.

CHAPTER 5

Experiments and Discussion

5.1 Assembling the Dataset

In the piano score-following problem, the dataset should include three parts: first, the digital musical scores that represent a piece as a sequence of chords (described in Section 2.1.1); second, the performance audio recordings of those pieces; third, the chord onset times in the performance audio—our *ground truth*. Some sites on the Internet provide MIDI files that contain the score information. Examples of such sites are musescore.com, kunstderfuge.com, and piano-midi.de. From a MIDI score file, we can extract the notes along with their score positions, and arrange these notes/chords in chronological order as in Table 2.1. I collected the performance recordings for my experiments from three sources: the Internet (e.g., imslp.org), the MAESTRO dataset (described below), and a few recorded performances by two pianists at Indiana University Bloomington (IUB). All performances were recorded on acoustic pianos.

The chord onset times in the performance audio are usually difficult to obtain without tedious work of hand labeling or correction. One possible approach is to get "close to perfect" alignment results from an offline audio-to-score alignment program (an easier task as described in Section 1.2.1), and then manually correct the results. Another, possibly better, approach is to record the performances on pianos that can capture movements of the keys, hammers and pedals, and store the performance information in MIDI files (e.g., on *Disklavier* pianos). Different from the MIDI score files, these MIDI *performance* files tell us what notes are being played at what time in real piano audio recordings, following a consistent order indicated in the score. With this information, we can easily match every score position with a played note, and thus with the specific time stamp of that

note in the performance. Even with occasional mistakes in the performance (wrong notes, extra/missed notes), we can still infer the matching between score positions and played notes by computing the *minimum edit distances*. The MAESTRO dataset by Hawthorne et al. (2018) contains such audio and MIDI data of 172 hours of piano performances from the International Piano-e-Competition[1].

My experiments contained 50 excerpts of real performances from 15 solo piano pieces, as shown in Table 5.1. The pieces are chosen in an *ad hoc* way, but most of them are from the Romantic era, and therefore tend to have abundant tempo variation within a piece (Kamenetsky et al., 1997), and probably involve heavy pedaling. Each excerpt's measure range and audio length is presented in Table 5.3. A typical excerpt lasts from 40 to 90 seconds, making the entire data set 48 minutes of music. The excerpts were cut out from full performances such that the style (e.g., fast or slow) is relatively consistent throughout an excerpt. About 33 minutes of the audio data was from the MAESTRO, while the other 15 minutes came either from the publicly available music recordings on the Internet or from the two pianists at IUB. For the first set of excerpts, I matched the performance MIDI data with the digital score according to the minimum-edit-distances criterion, with some minor manual corrections. For the latter two sets, I ran an offline audio-to-score algorithm which generated "close to perfect" results, and then manually corrected them. When the offline audio-to-score program failed occasionally (a large portion of the audio being misaligned), it could still be recovered by manually fixing some critical onsets, thus constraining the search space, before running the program again.

5.2 Evaluation Method

At any given audio frame of an excerpt, a score-following program holds a "belief" about where the performance is in the score, expressed by the *filtered probabilities* of different score positions: the probability of a hidden state given the observed data up until the

[1] http://piano-e-competition.com/

Composer	Piece	Abbreviation	#Excerpts
Mozart	Piano Concerto No. 17 in G major, mvmt1	Mozart	3
Schumann	Piano Concerto in A minor, mvmt1	Schumann	3
Chopin	Barcarolle, Op. 60	Chopin_barcarolle	2
Chopin	Prelude, Op. 28 No. 4	Chopin_prelude	2
Chopin	Ballade No. 1	Chopin_ballade	8
Liszt	Paganini S.141, No. 3 (*La campanella*)	Liszt	5
Rachmaninoff	Prelude, Op. 3, No. 2	Rachmaninoff	5
Schubert	Six Moments, D. 780 No. 2	Schubert_780	1
Schubert	Ständchen, D 957 No. 4 from Schwanengesang	Schubert_ständchen	4
Debussy	Prelude, No. 2 (*Violes*)	Debussy_violes	1
Debussy	Prelude, No. 8 (*La fille aux cheveux de lin*)	Debussy_fille	3
Beethoven	Piano Sonata No. 8 (*Sonata Pathétique*)	Beethoven_pathétique	1
Beethoven	Piano Sonata No. 31	Beethoven_31	8
Haydn	Piano Sonata No. 24 in D major, mvmt1	Haydn_mvmt1	1
Haydn	Piano Sonata No. 24 in D major, mvmt2&3	Haydn_mvmt23	3

Table 5.1: Piano music used in the experiments.

current moment. There might be multiple filtered probabilities of the same score position, but at different micro states (the baseline, and the timbre tracking model in Chapter 3), or with different ages (the tempo tracking model in Chapter 4). Naturally, the probability of recognizing the true note at a particular frame is the sum of the filtered probabilities that are associated with the *correct* score position.

Let $\kappa_1, \ldots, \kappa_N$ denote the ground truth index labels of all audio frames in an excerpt. For example, if $\kappa_n = m$, it means that the nth frame is in the mth note. Equation 5.1 presents the accuracy at frame n for the baseline and the timbre tracking models, where the filtered probabilities are defined in Section 2.4.3 and Section 3.4 respectively, and $C(\cdot)$ is a function that returns the corresponding note index of a given micro state, as defined in Section 3.3.2.

$$Acc_n = \sum_{x_n : C(x_n) = \kappa_n} p(x_n | y_{1:n}) \tag{5.1}$$

Equation 5.2 presents the accuracy at frame n for the tempo tracking model, where the

filtered probabilities are defined in Equation 4.13.

$$Acc_n = \sum_{\substack{k_n=\kappa_n \\ 1\le a_n\le n-\kappa_n+1}} p(k_n, a_n|y_{1:n}) \tag{5.2}$$

The above two equations measure the percentage of the belief held by the program that's correct about the score position at the nth frame. The overall accuracy on an excerpt is the average of those percentages across all frames:

$$Acc = \frac{1}{N}\sum_{n=1}^{N} Acc_n$$

This new evaluation method is called *frame-wise accuracy* (Jiang and Raphael, 2019), accounting for the accuracy of every frame. It is the first evaluation method that successfully avoids the trade-off issue between accuracy and promptness. (As in the offline-alignment problem, existing online evaluation methods for the score-following problem use *hard labels*, meaning every audio frame is assigned with only one score position, as in Cont et al. (2007). Therefore, with those evaluation methods, a program reports note onsets (the frames where new notes start), and the labels of the frames in between onsets can be naturally inferred. However, there is an intrinsic tendency for a program to delay reporting a new onset, because the program needs to receive some amount of data as evidence before it can be certain enough to report an emerging new note. Any program that reports hard onsets involves this design issue: how certain is certain enough? Longer delay allows the program to observe more data, and thus leads to higher certainty, but it sacrifices promptness, which is an important aspect for many applications based on score following (example applications are mentioned in Section 1.2.2). The delayed detection is illustrated in Figure 5.1.) The frame-wise accuracy evaluation method cleverly avoids this issue by allowing programs to report *soft* labels for every frame, which can include multiple score positions, each associated with a probability to represent certainty.

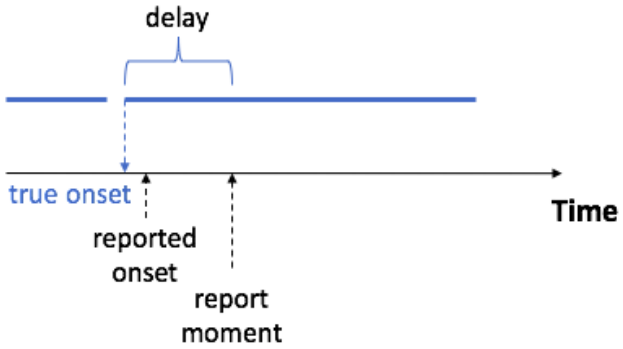


Figure 5.1: Illustration of delayed detection: longer delay leads to more accurate detection.

5.3 Experiment Plan and Settings

To test the two proposed methods, I ran two sets of experiments. The first set aims to compare the timbre tracking algorithm with the baseline, and the second aims to compare the tempo tracking algorithm with the baseline. The baseline program, based on the hidden Markov model discussed in Section 2.4, is a state-of-the-art program. This original baseline program uses a sampling rate of 8 kHz, a frame length of 512 samples, and a hop size of 256 samples. Both sets of experiments share the same settings as the original baseline, except that the hop size in the second set is decreased to 128 samples. This is because the tempo tracking algorithm needs a finer temporal resolution in order to reasonably approximate the note-wise representation with the frame-wise representation, as discussed in Section 4.2. The hop sizes used by the programs in both sets are specified in Table 5.2.

	new method	**baseline**
Set 1	256 (*32 ms*)	256 (*32 ms*)
Set 2	128 (*16 ms*)	128 (*16 ms*)

Table 5.2: Hop sizes in the two sets of experiments (*ms* = *milliseconds*).

Every piece has a *tempo marking* on the first page of the musical score, which indicates a suggested tempo range for the piece. For example, the tempo marking *Andante* usually corresponds with 76-108 beats per minute (bpm). I set the default tempo of a piece to the middle of the suggested range, so 92 bpm for all *Andante* pieces. This value is used to

construct the state graph discussed in Section 2.4.1, where the number of micro states and self-loop probabilities of a note are decided by the nominal duration (in beats) of this note and the default tempo. This value is also used for deciding the initial tempo in the tempo tracking method in Chapter 4. The concept of "tempo" there is defined as the number of seconds spent in a whole note, which can be converted from bpm given the time signature. For example, in a piece at $\frac{3}{4}$ time, *Andante* has the tempo value of 2.6 ($= 4 \times 60/92$) in the tempo tracking method. Sometimes, there are resetting tempo markings in the middle of a piece, and the program gets a new default tempo starting from those places; however, all localized tempo directions in the score that deviate from the default are ignored, because they are subjective and can vary widely from performance to performance.

There are five important parameters in the timbre tracking algorithm: the decay factor (λ), the variance of the process noise (σ^2), the variance of the observation noise (ρ^2), and the initial mean and variance of the partials' amplitudes when a new partial comes into existence (m_0 and v_0). These were manually set in the experiments.

5.3.1 Tempo Variance

The most important parameters in the tempo tracking algorithm are the variances of the two noise terms: the noise in the onset (ε_{k+1}), and the noise in the tempo (η_{k+1}).

$$o_{k+1} = o_k + l_k t_k + \varepsilon_{k+1}$$

$$t_{k+1} = t_k + \eta_{k+1}$$

In Chapter 4, these are all 0-mean Gaussian noise terms with fixed variances, and all mutually independent. However, the statistics on this dataset suggest that these assumptions are incorrect. I have revised these assumptions as I discuss later in this section.

From the ground truth, I calculated the mean and standard deviation of the local tempo for each of the 50 excerpts, and generated a scatter plot in Figure 5.2, in which each data point represents an excerpt. The tempo is defined as the number of seconds spent on a

whole note, so a larger value means slower tempo; a local tempo value of a chord can be calculated by dividing the actual time spent on this chord by its nominal musical length. The tempo mean and variance of an excerpt are based on all local tempo values. The plot indicates a positive linear relationship between the two, and their correlation is calculated as 0.45.

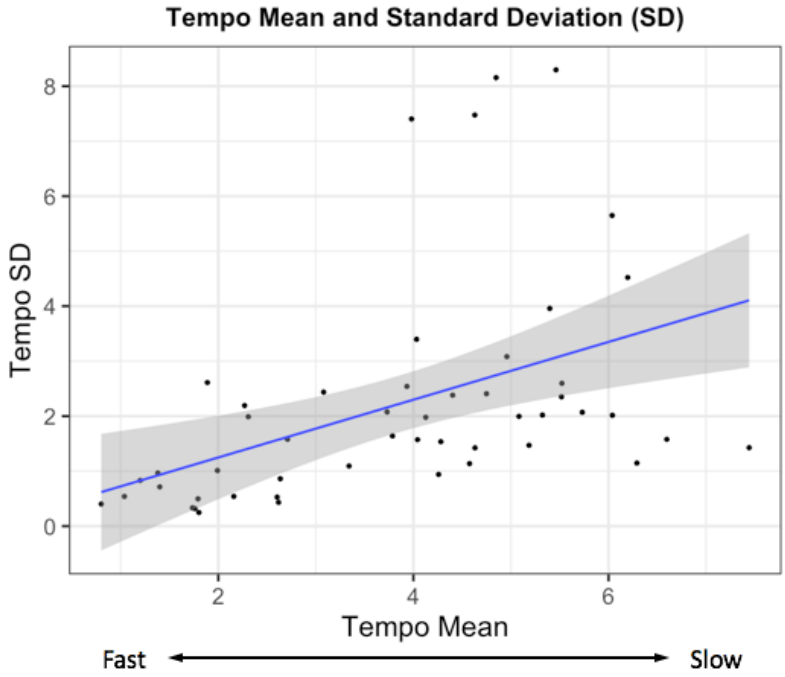


Figure 5.2: Relationship between mean and standard deviation of the tempo, indicated by the 50 excerpts. Grey area represents the 95% confidence interval.

When plotting the tempo mean with the *relative* standard deviation (standard deviation/mean), however, the line is almost flat as in Figure 5.3, and the correlation between the two is 0.03, which is quite low. This indicates that the standard deviation of the tempo is roughly proportional to the tempo, which is consistent with Weber's law (Fechner et al., 1966). If we think about a fast Mozart sonata and a slow Debussy prelude, we can understand why slower tempo (larger value) tends to have larger variance.

To account for this, I modified the noise term from the original model such that the standard deviation of the tempo noise, η_{k+1}, is proportional to the tempo value, and the resulting equation is no longer a standard linear dynamical model. With similar logic,

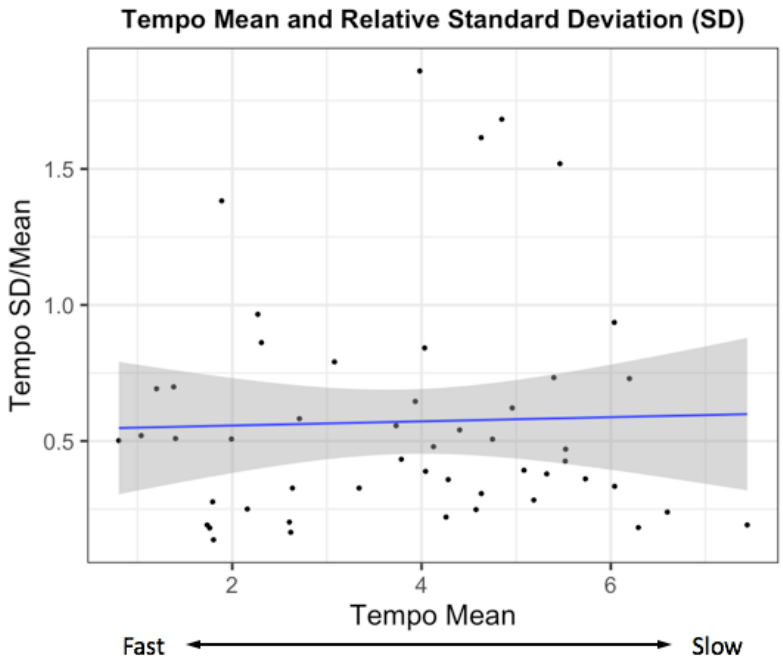


Figure 5.3: Relationship between mean and *relative* standard deviation of the tempo, indicated by the 50 excerpts. Grey area represents the 95% confidence interval.

the standard deviation of the onset noise term, ε_{k+1}, is also set to be proportional to the predicted note length, $l_k t_k$, because longer notes tend to have more room to deviate from their predicted onsets.

It's worth mentioning that the noise term in the onset, ε_{k+1}, is related to the *agogic accent* in music theory, which often refers to either 1) the extended duration of a note without changing the tempo, 2) temporarily slowing down the tempo, or 3) delayed onset of a note by doing a pause before starting the note. Therefore, when ε_{k+1} is positive, it can be viewed as the agogic accent.

5.3.2 Beam Search

To further reduce the computational complexity, I applied the *beam search* technique in all three algorithms. At every frame, the program ranks all hypotheses by their probabilities, and it keeps no more than a fixed number of hypotheses at the top, and discards the rest with lower probabilities. The maximum number of hypotheses the program can keep at any frame

is called the *beam width*. Of course, the program might prune out some correct hypotheses, but this is expected to happen rarely, and when it happens, the program should be able to recover in later frames (with no fatal error propagation). The beam width used in the experiments was 200, which is wide enough to be optimal; I conducted tests demonstrating that having a larger beam width than 200 does not improve the results noticeably.

5.4 Experimental Results

The two sets of experiments include four algorithms applied to each of the 50 excerpts respectively. The resulting 200 accuracies are reported in the last four columns of Table 5.3. The first set of experiments examines the *timbre* tracking algorithm by comparing the results with the baseline algorithm (the fifth and sixth columns). The second set of experiments examines the *tempo* tracking algorithm by comparing the results with the baseline algorithm (the last two columns). Piano music is one of the most challenging cases in the score-following realm, as these experiments demonstrated. For example, the state-of-the-art program is well established and has been refined over the past two decades to fail rarely on monophonic music, yet it failed to follow 30% of the piano excerpts here (the fifth column). There are various possible reasons that can make piano music an especially difficult case, as discussed in Section 1.4.1, and in more detail later through the case-by-case analysis.

I set the threshold for "very low" accuracy at 40%; it is, admittedly, a somewhat arbitary number, but the lowest accuracy level that still seems to be serviceable for many applications (as mentioned in Section 1.2.2) that depend on the result of score following. Any accuracy that's lower than 40% is marked in grey in the table, and we can regard these as *unsuccessfully followed*.

Very low accuracy can exemplify a fatal error, in which the program got lost at certain frames, and never found its way back. For example, in Excerpt #15 (Chopin's *Ballade No. 1*, measures ranging from 130 to 166), both programs in the first set had near zero

Index	Piece	Measures	*Length (sec.)*	Set 1 (%)		Set 2 (%)	
				Baseline	Timbre	Baseline	Tempo
1	Mozart	74-95	*36.5*	78.5	81.8	75.9	74.9
2	Mozart	139-172	*57.7*	82.3	81.2	77.6	78.7
3	Mozart	184-207	*42.2*	69.5	71.5	66.8	64.8
4	Schumann	1-5	*6.7*	70.8	79.3	78.5	63.7
5	Schumann	11-19	*20*	79.1	87.4	82.6	79.9
6	Schumann	58-67	*27.4*	76.4	80.8	79.2	83.5
7	Chopin_barcarolle	1-9	*42.8*	71.2	71.6	75.8	79.3
8	Chopin_barcarolle	1-9	*41.7*	66.3	66.7	61.3	57.4
9	Chopin_prelude	0-12	*67.3*	62.7	56.4	65.2	70.5
10	Chopin_prelude	13-26	*93.3*	48.5	50.5	52.3	52.5
11	Chopin_ballade	1-8	*40*	63.5	59.9	73.6	74.2
12	Chopin_ballade	8-35	*88*	65.4	66.3	68.7	68.8
13	Chopin_ballade	36-65	*49*	9.81	10.3	11.8	27.5
14	Chopin_ballade	65-93	*85*	59.1	55.8	50.4	63.0
15	Chopin_ballade	130-166	*44.1*	0.3	0.3	0.4	27.2
16	Chopin_ballade	166-193	*58.5*	0.3	0.3	0.3	36.2
17	Chopin_ballade	194-250	*85.2*	45.3	46.5	51.5	46.5
18	Chopin_ballade	250-262	*38.8*	15.6	8.6	31.2	15.5
19	Liszt	0-19	*43*	2.2	2.0	2.1	18.6
20	Liszt	20-40	*40.6*	74.7	71.3	71.4	76.1
21	Liszt	40-58	*36.4*	46.9	37.7	0.5	13.4
22	Liszt	59-86	*53*	46.8	37.4	48.2	25.1
23	Liszt	103-147	*74.5*	41.3	36.8	25.7	51.2
24	Rachmaninoff	1-9	*48*	80.4	79.1	84.4	88.5
25	Rachmaninoff	9-14	*45*	72.6	59.5	80.9	86.3
26	Rachmaninoff	15-43	*41.3*	21.0	13.1	24.2	25.3
27	Rachmaninoff	44-55	*55.7*	65.8	64.9	69.1	71.8
28	Rachmaninoff	55-62	*41.9*	55.0	42.1	42.7	69.4
29	Schubert_780	18-36	*51.4*	72.3	71.8	75.1	82.5
30	Schubert_ständchen	1-37	*102*	60.5	63.3	56.2	55.6
31	Schubert_ständchen	38-70	*90.3*	56.0	58.6	49.9	53.2
32	Schubert_ständchen	71-102	*85.4*	52.4	49.5	48.6	49.5
33	Schubert_ständchen	103-115	*62.8*	25.1	5.0	4.1	10.1
34	Debussy_violes	0-21	*56.4*	63.8	68.1	69.1	73.6
35	Debussy_fille	1-18	*71.9*	72.3	70.1	76.6	86.6
36	Debussy_fille	19-38	*78*	61.8	54.0	65.0	70.4
37	Debussy_fille	1-19	*87.6*	70.1	70.0	79.2	79.2
38	Beethoven_pathétique	1-10	*93.3*	71.8	68.1	76.4	82.4
39	Beethoven_31	1-11	*41.2*	30.9	46.4	52.6	50.5
40	Beethoven_31	12-25	*42.5*	31.0	47.1	49.4	63.3
41	Beethoven_31	25-39	*52.8*	26.9	29.1	48.8	56.1
42	Beethoven_31	40-55	*51.5*	25.1	33.6	48.2	65.6
43	Beethoven_31	56-75	*65.7*	13.2	25.2	31.8	46.5
44	Beethoven_31	76-90	*51.8*	29.7	34.5	47.0	69.1
45	Beethoven_31	90-104	*49.5*	38.7	18.7	52.8	80.4
46	Beethoven_31	105-116	*44.9*	16.3	22.4	34.0	62.7
47	Haydn_mvmt1	0-51	*102*	56.2	48.3	50.7	50.7
48	Haydn_mvmt23	1-8	*90.3*	78.4	86.0	83.1	85.5
49	Haydn_mvmt23	9-24	*85.4*	61.5	62.8	64.9	46.9
50	Haydn_mvmt23	25-62	*62.8*	62.6	64.2	69.3	73.3

Table 5.3: Experiment results on 50 excerpts, and measures and lengths of these excerpts.

accuracy at any given frame, and failed to make any meaningful judgement about where the performance was in the score. Very low accuracy can also mean high uncertainty—the program is correct about what *region* the performance is at in the score (e.g., the correct measure), but it is not certain about which exact chord is sounding in that region, probably because of similar spectral features among neighboring chords. This subset of very low accuracy is not an absolute loss; it still contains valid information that may be useful for some applications (e.g., a page turner), although not for the purposes under consideration here. To give an example, in Excerpt #33 (Schubert's *ständchen*, measures ranging from 103 to 115), the baseline program was just 25% sure at any given frame on average, but the mistakes were within the "safe range" to avoid fatal errors. Figure 5.4 shows a typical scenario of such mistakes where the program was "confused" among neighboring chords, but was right about the general range. This is a snapshot of all hypotheses along with their probabilities at the 940th frame; this frame is chosen to be in the middle of the 929th chord in the score, which corresponds to the fourth chord in the 110th measure, as shown in Figure 5.5. The true score position was "110+3/8", but the program gave more weight to the previous three chords in this measure. From the musical score shown in Figure 5.5, we can see that the fourth chord shares three notes with the second chord, which makes them "sound similar" to the program. The musical performance of this measure also involves pedaling, which made these chords even more difficult to distinguish from each other. Both phenomena can be observed in the spectrogram (spectrums in a range of time) of this measure, as shown in Figure 5.6.

In the fifth column, where the results correspond with the original baseline program, all eight excerpts in *Beethoven Piano Sonata No. 31* got very low accuracy, although not getting completely lost. I generated a "movie" version of Figure 5.4, which can show the hypothesis distribution one-by-one quickly through all frames of an excerpt, and used it to inspect the nature of errors in these excerpts. None of these eight excerpts contains fatal errors; the program was mostly right about the region in the score, but was confused by

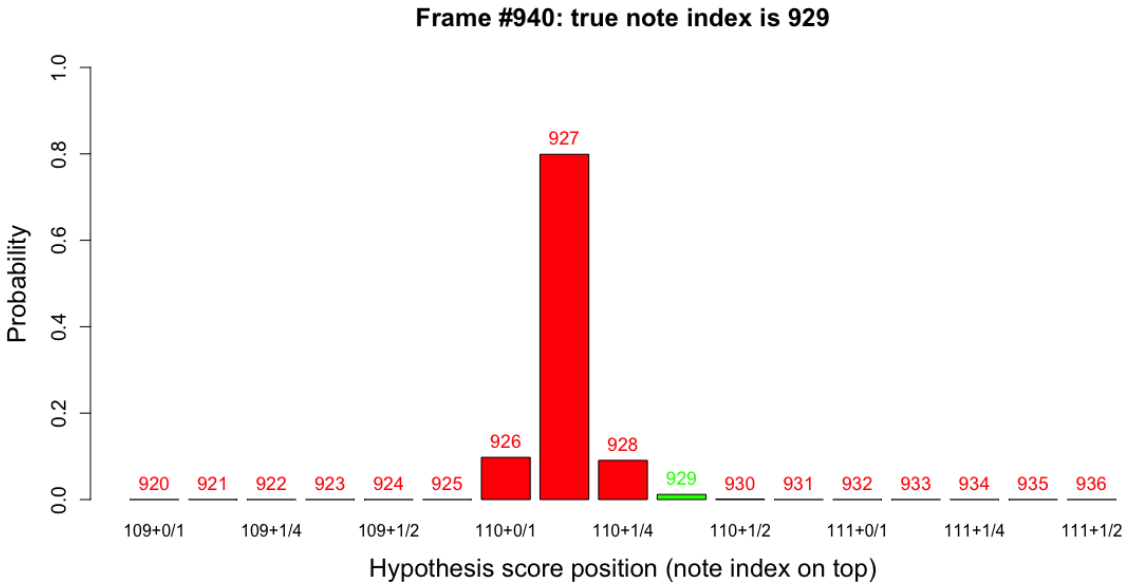


Figure 5.4: The probability distribution among all hypotheses at the 940th frame in Excerpt #33. The true note index is marked as green; other hypotheses are marked as red.

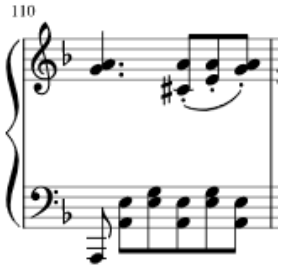


Figure 5.5: 110th measure of Schubert's *ständchen*, which includes six chords (musical events). (from musescore.com)

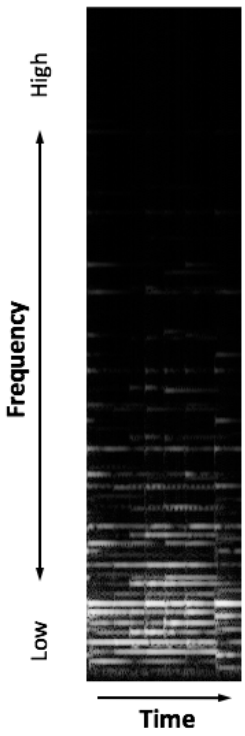


Figure 5.6: The spectrogram of the 110th measure of Schubert's *ständchen*. Brighter pixels represent higher values.

neighboring chords. In fact, *most of the time*, the program mistakenly gave *later* chords higher probabilities than the correct chord. Figure 5.7 shows an example of this kind of premature detection (as opposed to delayed detection in Figure 5.4). This indicates a systematic error that makes the program expect faster performance than the given data—an incorrect timing model. Indeed, although the tempo marking of this piece is *Moderato* (about 105 bpm), musicians usually play it at a much slower pace (about twice as slow). I conducted the experiments again on these eight excerpts adjusted with a lower default tempo, and the baseline program was able to follow all excerpts with dramatically higher accuracy, as reported in Table 5.4. This confirms that a correct timing model is vital for score following. In contrast, even with an incorrect initial tempo value, the tempo tracking algorithm was able to follow all eight excerpts (the last column in Table 5.3), with an average accuracy of 61.8%. The tempo tracking algorithm is designed to be able to adapt to the tempo indicated from the data, and thus is able to recover from a falsely assigned intial

tempo value and form a more accurate timing model.

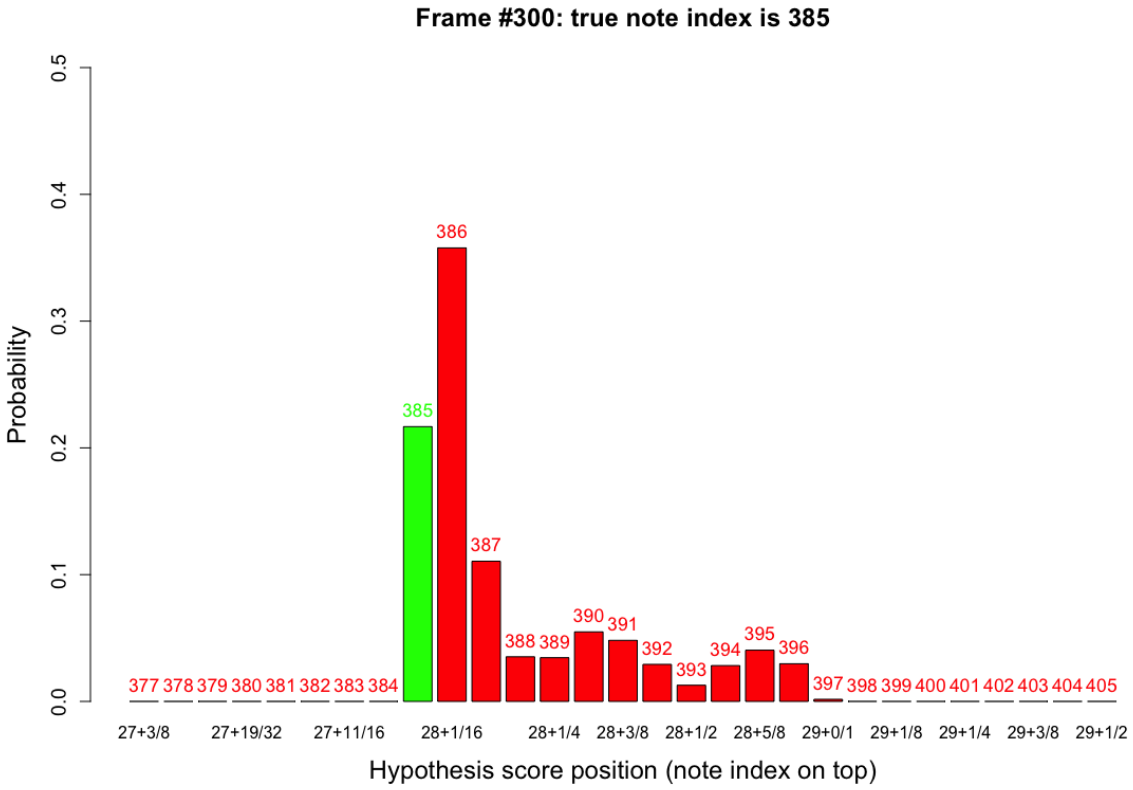


Figure 5.7: An example of premature detection (from Excerpt #41).

Excerpt Index	**Accuracy (%)**	
	105 bpm	55 bpm
39	30.9	73.7
40	31.0	67.8
41	26.9	68.9
42	25.1	66.2
43	13.2	54.5
44	29.7	72.0
45	38.7	80.3
46	16.3	58.4
average accuracy	26.5	67.7

Table 5.4: Accuracy using different default tempo values in the baseline.

Beyond the eight excerpts in *Beethoven Piano Sonata No. 31*, the original baseline program also failed to follow seven other excerpts. I examined each of them using the

"movie" tool, and by listening to the audio recording. Four of them are from Chopin's *Ballade No. 1*: Excerpts #13, #15, #16, and #18. These four excerpts are dramatically fast, in contrast to other followed excerpts of the same piece. The program either got lost near the beginning (Excerpts #15 and #16), or followed successfully part-way through the performance until the sound started to blur because of rapid playing with pedal (at the 42nd measure in Excerpt #13 and at the 253rd measure in Excerpt #18). This indicates that the program failed in those excerpts probably because of an unreliable data model. This "blurring" effect usually happens when a sequence of chords is played in a rapid fashion, oftentimes with some degree of pedaling, making it difficult to hear individual notes. When the sound starts to blur, the data model cannot offer as much information to distinguish neighboring chords, because different hypotheses end up with similar likelihoods; the program, therefore, relies more on the timing model, which could be incorrectly assumed in the baseline algorithm. In contrast, three out of these four excerpts were followed with noticeably higher accuracy by the tempo tracking algorithm, although still under 40%. In the tempo tracking algorithm, the program assumes the tempo to be steady, until the data suggest otherwise; when the data likelihoods are similar, the program assumes the performance is proceeding with the previous tempo—the most likely case in reality. Therefore, the tempo tracking algorithm should be more robust than the baseline, especially when the data model is unreliable. Since it needs multiple notes to create the "blurring" effect, this challenge does not exist in monophonic music.

In excerpt #19, the baseline program suffered from delayed detection within the first few seconds, and got lost for the rest of the performance. Figure 5.8 shows the musical score of the beginning of this excerpt. All notes in the first four measures are D from different octaves, and the score contains repeated chords and patterns. Figure 5.9 shows that the harmonics from the lower chords extended to the higher chords, which caused delay in detecting the higher chords—this issue does not exist in monophonic music. Because of the repeated chord pattern, the program also mistook later measures for earlier ones.

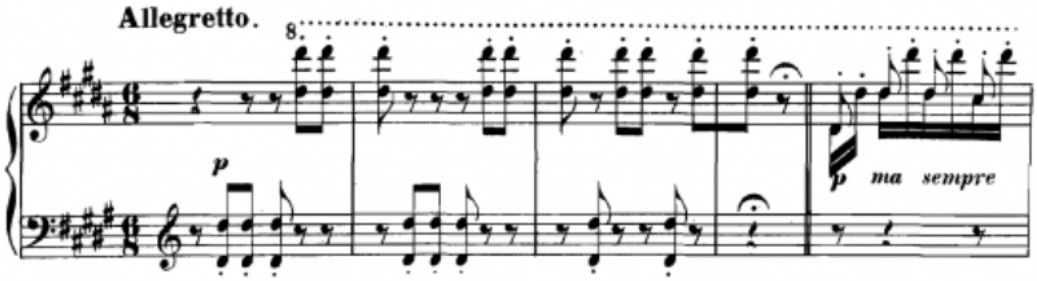


Figure 5.8: Beginning of Liszt's *La campanella*. (from imslp.org)

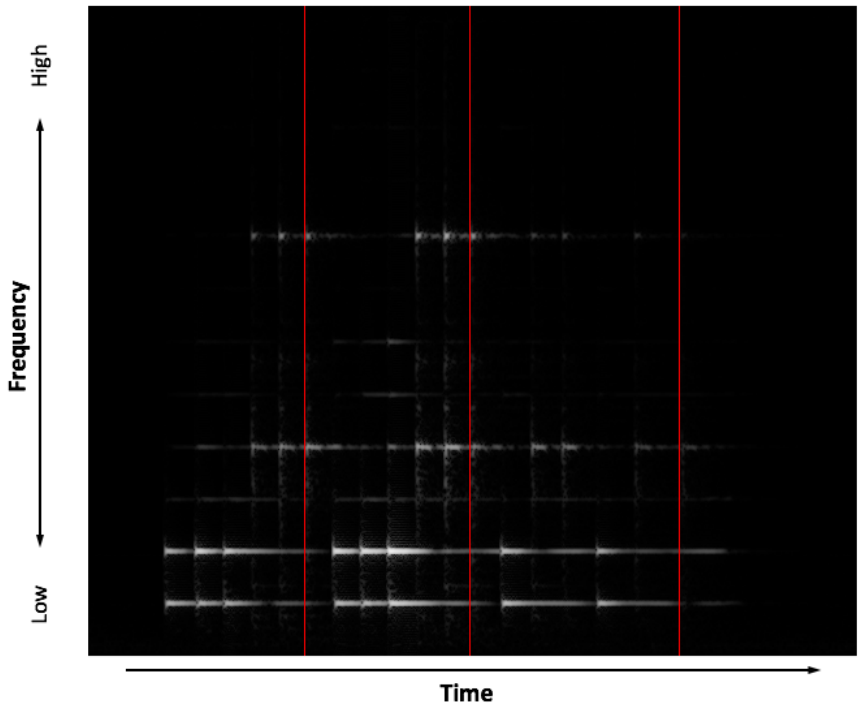


Figure 5.9: The spectrogram of the first four measures of Liszt's *La campanella*. Red lines are the onsets of the first chord in each measure.

For the other two failed excerpts, #26 and #33, the program didn't get completely lost, but rather was confused among neighboring chords. The excerpts' musical scores suggest that the neighboring chords oftentimes share common notes, and exhibit repeated chord patterns, which explains the program's uncertainty.

5.5 Timbre Tracking Algorithm Discussion

This section focuses on discussing the timbre tracking algorithm, as tested in the first set of experiments. Out of the 50 excerpts, 18 excerpts had very low accuracy (< 40%) by at least one of the two algorithms (the baseline or the timbre tracking algorithm). Table 5.5 shows the count of such low-accuracy excerpts for each of the two algorithms, and the average accuracy of those excerpts. The timbre tracking algorithm got about as many low-accuracy results as the baseline algorithm (exactly one more instance); the average accuracies of these excerpts are also comparable, with only 0.6% difference.

	baseline	**timbre tracking**
# failed excerpts	15	16
average accuracy	19.1%	19.7%

Table 5.5: Counts of low-accuracy excerpts for both algorithms, and their average accuracies.

Excluding those 18 excerpts, I calculated the average overall accuracy across the other 32 excerpts that both algorithms successfully followed, as shown in Table 5.6. The timbre tracking algorithm has comparable results with the baseline, with only 0.3% difference.

	baseline	**timbre tracking**
average accuracy	65.6%	65.9%

Table 5.6: Average accuracies of 32 excerpts.

The timbre tracking algorithm did not show better results than the baseline algorithm, in terms of either lost frequency or average accuracy. Out of the 15 excerpts failed by the

baseline, 13 of them were also failed by the timbre tracking algorithm. This implies that there were some challenging aspects the new method failed to address or improve.

The timbre tracking algorithm is designed to provide a better data model than the baseline by allowing the chord templates to adapt to the given data through a series of Kalman filters. I argue that, in theory, the new method can provide a more discriminating data model for one key reason. Both correct and incorrect hypotheses can score better by adapting their templates to the data—however, the correct hypotheses have greater potential to adapt *well*, because their templates have the *right* adapting freedom that incorrect hypotheses don't necessarily have. For example, an incorrect hypothesis has to ignore an observed peak because its template lacks the corresponding harmonic of this peak.

In practice, however, the Kalman filters need a certain number of updating iterations before they can adapt well to the observed harmonic amplitudes, especially when the initially set amplitudes are quite different from the truth. However, a chord only lasts 8.45 frames (0.27 second) on average among the 50 excerpts. For short chords in a fast piece, it is possible that the program has to move on to the next chord before it can develop an accurate template. Therefore, the timbre tracking algorithm should show its advantage more in later frames than in the first few frames. To investigate this, I excluded the first six frames (about 0.2 second) of all chords, and recalculated the frame-wise accuracy for each excerpt. The timbre tracking algorithm achieved 2.1% higher accuracy on average, as shown in Table 5.7.

	baseline	**timbre tracking**
average accuracy	66.7%	68.8%

Table 5.7: Average accuracies of 32 excerpts, with first six frames excluded.

To confirm that this improvement is statistically significant, I conducted a *paired sample t-test* among the 32 excerpts. The null hypothesis assumes that the true mean difference

between a pair of accuracies from the baseline and the new method is zero:

$$H_0 : \mu_d = 0$$

where μ_d is the population mean matched-pair difference. The other coefficients are listed in Table 5.8. The p-value of 0.06217 shows that these results are significant at the $\alpha = 0.1$ level; it indicates that the new method produces measurably better results than the baseline in later frames.

coefficient	**value**
$\bar{d}$	-0.021
t	-1.9349
df	31
p-value	0.06217

Table 5.8: Paired t-test to compare timbre tracking algorithm and baseline, with first six frames excluded.

Because the timbre tracking method tends to show more advantage in later frames, we can imagine an algorithm to work just as well or even better if it uses the fixed templates from the baseline for the first few frames, and uses flexible templates for later frames. (However, since piano is a percussive instrument, the first couple frames of a chord might not exemplify meaningful harmonic features, and therefore the spectral features might not be ideal under either method.) A 2.1% improvement in later frames is a positive sign of an improved data model; however, the remaining high failure rate (16/50) indicates that this approach has not addressed the most urgent issues in the baseline.

One can speculate that perhaps the data model does not need to be "perfect": humans can still follow a very old music recording with low sound quality or with some level of noise in the background, without much more difficulty than following a live performance. Perhaps the fixed data model in the baseline is already good enough, and the room for further improvement is limited; especially in places where the sound is blurring and the neighboring chords sound similar (some of the most challenging cases), the data model

tends to give different hypotheses similar weights anyway, and thus cannot offer much meaningful information. On the other hand, the data model can include other features besides the spectrum, which are not investigated here; for example, the total energy from all frequencies, and spectrum differences between neighboring frames. The idea of tracking the time-changing nature of a chord can be generalized to these other features. The timbre tracking method represents a flexible framework that can incorporate multiple features, working synergistically to provide a more accurate and discriminating data model from different perspectives.

5.6 Tempo Tracking Algorithm Discussion

The second set of the experiments examines the tempo tracking algorithm, comparing its results with those of the baseline algorithm (the last two columns in Table 5.3). Out of the 50 excerpts, 12 excerpts had very low accuracy with at least one of the two algorithms. Table 5.9 shows the number of such low-accuracy excerpts, and the average accuracy of those excerpts. The tempo tracking algorithm failed exactly two times fewer than the baseline, and it also had higher average accuracy among the failed excerpts.

	baseline	**tempo tracking**
# failed excerpts	11	9
average accuracy	15.1%	22.1%

Table 5.9: Counts of low-accuracy excerpts for both algorithms, and their average accuracies.

Excluding those 12 excerpts, I calculated the average overall accuracy across the other 38 excerpts that both algorithms followed, as shown in Table 5.10. The tempo tracking algorithm achieved about 4.1% higher accuracy than the baseline.

	baseline	**tempo tracking**
average accuracy	65.0%	69.1%

Table 5.10: Average accuracies of 38 excerpts.

To verify that this accuracy difference is statistically significant, I compared the two algorithms by conducting a paired samples t-test. The null hypothesis assumes zero mean difference between a pair of accuracies from the baseline and the new method:

$$H_0 : \mu_d = 0$$

where μ_d is the population mean matched-pair difference. The other coefficients are listed in Table 5.11. The low p-value (< 0.01) indicates that we can reject H_0, and that the new method is measurably better than the baseline.

coefficient	**value**
$\bar{d}$	-0.04
t	-2.7307
df	37
p-value	0.009623

Table 5.11: Paired t-test to compare tempo tracking algorithm and baseline.

The new method aims to provide a more accurate timing model than the baseline by allowing the tempo to adapt to the data. The tempo tracking algorithm treats the tempo as a variable, which can be adjusted to be higher or lower according to the observed data using a Kalman filter—a large onset gap suggests a low tempo and vice-versa. This way, the program can track the unpredictable timing variations in a performance. Modeling the tempo as smooth also helps discriminate between more and less likely hypotheses; for example, abrupt tempo changes are generally less likely. An accurate timing model is especially helpful when the data model is unreliable, as discussed in Section 5.4.

I used the "movie" tool mentioned in Section 5.4 to investigate each of the 12 excerpts failed by either algorithm, focusing on contrasting the new method with the baseline. Three out of the 11 excerpts failed by the baseline were successfully followed by the tempo tracking algorithm: Excerpt #23 from Liszt's *La campanella*, and Excerpts #43 and #46 from Beethoven's *Piano Sonata No. 31*. As was already discussed in Section 5.4, the latter piece was set with a default tempo value far from reality, but the tempo tracking algorithm

was able to correct this value by tracking the tempo using Kalman filters. Among the other six excerpts of this piece that were followed by both algorithms, the tempo tracking algorithm also achieved 14.4% higher accuracy on average—a dramatic improvement. For the other excerpt from Liszt's *La campanella*, the baseline program got completely lost half-way through after a section of 14 repeated chords in a row, while the tempo tracking algorithm followed this excerpt without a fatal error.

Several excerpts, although followed with low accuracy by both methods, nevertheless provided evidence that the new method is more robust than the baseline. The baseline got completely lost near the beginning of Excerpts #15 and #16 where the sound was blurring, while the tempo tracking algorithm followed the region (although it failed to distinguish neighboring chords well). In Excerpt #13, the baseline got lost starting around Measure #13 (about 1/3 through the excerpt) when the sound started to blur, and never recovered; in contrast, although the tempo tracking algorithm was sometimes confused by the neighboring measures, it always recovered. Similar observations exist in Excerpts #19 and #21: both algorithms got lost near the beginning of these excerpts, probably because of repeated chords and patterns, but only the tempo tracking algorithm recovered afterwards. All these cases suggest that a flexible timing model tends to recover from mistakes easier than the fixed timing model. I speculate that this is because the tempo tracking algorithm can reach faraway states more *quickly* than the baseline. For example, to recover from a mistake that's 10 chords away, the baseline needs at least $10 \times n$ frames to go through all the micro states in the state graph, assuming n is the average number of micro states for a chord. The tempo tracking algorithm, however, can reach a 10-chord distance within only a small number of frames (possibly 10 frames) by allowing the tempo to be fast.

In Excerpt #33, the tempo tracking algorithm got completely lost after Measure #106, which includes a sequence of 22 very fast notes of equal nominal length, as shown in Figure 5.10. In the performance, however, these notes' lengths vary tremendously, especially at the two ends, as shown in Figure 5.11. The tempo tracking algorithm was able to track

the tempo and follow the notes at the beginning of this sequence, but failed to adapt to the "sudden" slowing down at the end: the tempo value tripled at the last note (three times as slow). Although the dramatic slowing down does not feel sudden or surprising to humans—because the dynamics of the previous few notes in the performance strongly indicate the upcoming slowing down—it is considered to be an unlikely situation by the program. Machines are musically less intelligent than humans, at least in this case.

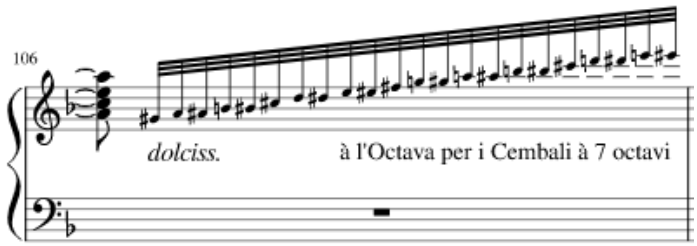


Figure 5.10: 106th measure of Schubert's *ständchen*. (from musescore.com)

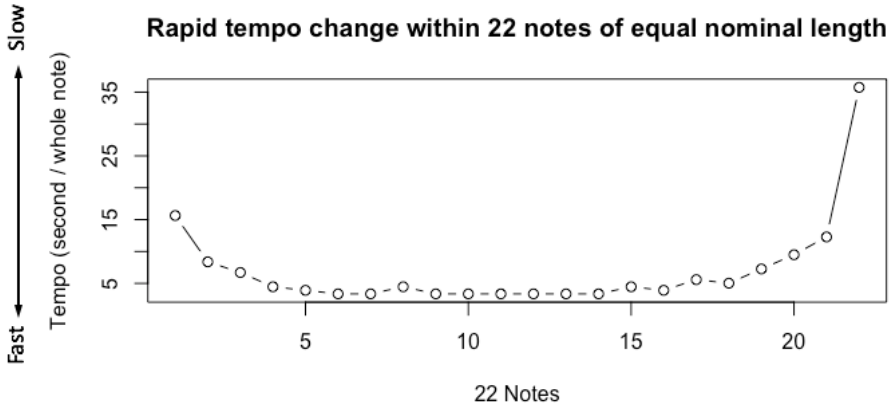


Figure 5.11: Rapid tempo change within 22 short notes.

Among the 12 excerpts where either algorithm failed, the tempo tracking algorithm achieved higher frame-wise accuracy than the baseline (including all the nine excerpts analyzed above) except for two: Excerpts #18 and #22. In Excerpt #18, the tempo tracking algorithm got lost at Measure #253, and falsely recognized the rest of the excerpt to be in this measure. Figure 5.12 shows the musical score of this region. Although the time signature is $\frac{2}{2}$, these 28 16th notes do not add up to 1. This is called "impossible rhythm" by Hook (2011) (as is the measure in Figure 5.10). The 253rd measure is near the end

of this piece, and can be viewed as a virtuosic passage at the climax of the piece, usually with great expressive freedom. Indeed, this measure is played at a faster tempo than other measures in the region, and each of these chords lasts only 4.26 frames (0.07 second) on average. If we treat them as "28th" notes, a minor error of one frame (16ms) for an onset could result in a difference of $0.016 \times 28 = 0.448$ (second per whole note) in the tempo value, which is considerable given that the average tempo value of this meansure is 1.908 (seconds per whole note). This also means that the "observed tempo" (calculated from the onsets) has to be a multiple of 0.448, and cannot be more precise. Furthermore, perhaps more importantly, because this measure is played quite fast, the sound also blurs (as in Figure 5.13), which makes it difficult for the data model to provide meaningful evidence about the tempo changes. We can speculate that both the coarse tempo observation and the blurring effect left the program unable to catch up with the high tempo and stuck at this measure longer than reality.

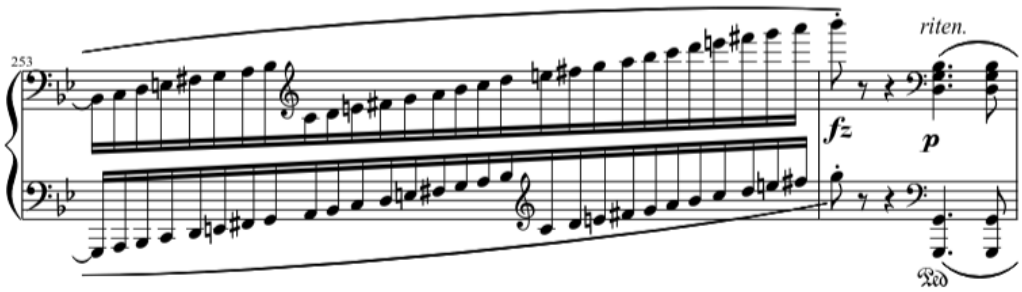


Figure 5.12: 253rd and 254th measure of Chopin's *Ballade No. 1*. (from musescore.com)

Excerpt #22 is the other failed excerpt followed with lower accuracy by the new method than by the baseline. The tempo tracking algorithm never got completely lost in this excerpt, but was highly uncertain among neighboring chords or measures. In particular, a chord was oftentimes mistaken for the preceding chords. That is because, according to the spectrogram of this excerpt, partials from previous chords are commonly extended to later chords by pedaling. This excerpt is also played the fastest among all the excerpts, with 11.7 chords played per second on average. Most measures of this excerpt only have 32nd notes, which means that a one-frame difference brings a $0.016 \times 32 = 0.512$ (second per whole note)

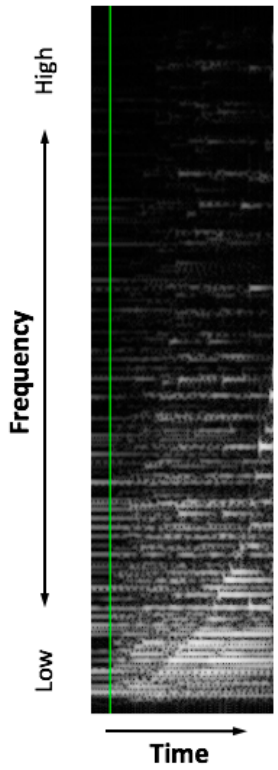


Figure 5.13: The spectrogram of the 253rd measure of Chopin's *Ballade No. 1*. The green line represents the first onset of this measure (the chord with the notes A-C).

difference in the tempo, and that the "observed tempo" has to be multiples of this value. By comparison, for a slower piece with mostly quarter or eighth notes, the resolution for observed tempo is $0.016 \times 4 = 0.064$ or $0.016 \times 8 = 0.128$, respectively. Low resolution in the observed tempo values in this excerpt might have limited the tempo tracking algorithm from tracking the tempo accurately, and the resulting timing model could possibly be even worse than the baseline. To test this, I doubled the resolution for this excerpt by halving the hop size to 8ms, and the new frame-wise accuracy indeed increased by 9.1%. However, as mentioned later in Section 5.7.1, the reasons why decreasing the hop size might help with score following cannot yet be precisely determined.

The remaining one of the 12 failed excerpts for analysis is Excerpt #26, which got comparable results from both algorithms. This is also a fast excerpt, with 8.4 chords played per second on average (the mean across all excerpts is 3.8). With pedaling and fast playing, there are obvious blurring effects in the sound. In this excerpt, a measure often contains chords that share identical notes, and sometimes contains repeated chords. This pattern and pedaling explain why the program's errors were mostly delayed detection, but within the distance of about one measure.

The experimental results show strong promise in the direction of improving the timing model by tracking the tempo: the new method gained better accuracy among followed excerpts; the program was robust even when the default tempo was set far away from reality; and when the program got lost because of some challenging measures, the tempo tracking algorithm recovered from mistakes more easily than the baseline. The flexible timing model assumes the tempo is mostly steady, but allows the tempo to fluctuate up and down guided by the data model: when the data indicate it takes longer to play a certain chord, the tempo value is more likely to be larger (meaning slower), and vice-versa. In contrast, the fixed timing model in the baseline assumes a constant tempo throughout the piece (an unlikely case in reality), regardless of what the data indicate. In situations where the data model is less reliable (e.g., repeated notes in neighboring chords, pedaling, blurring

sound), the baseline tends to get completely lost more often than the new method, possibly because the baseline lacks a correct timing model to fall back on when the guidance from the data model is not clear.

The tempo tracking algorithm, however, can suffer at fast playing places, which usually involve lots of short notes (e.g., 16th and 32nd). As discussed earlier, this probably happens because the real onsets have to be approximated as multiples of the hop size (16ms), and the approximation errors are relatively large for short notes, which can cause incorrect estimation for the tempo.

This new method is generalizable and can be applied to a variety of instruments, monophonic or polyphonic. In the application of automatic accompaniment systems, the tracked tempo trajectory of previous notes can be used for anticipating the next note more accurately, possibly by incorporating some more sophisticated musical behaviors. For example, for a sequence of fast ascending notes in Figure 5.10, the rate of tempo reduction for the last few notes might be exponential instead of linear. Tracking and extracting tempo information in music performances is also meaningful for performance analysis, especially for analyzing the timing aspects of a piece based on a diverse range of recordings.

5.7 Overall Discussion

This section discusses various aspects of the piano score-following problem, providing more insights for future exploration and improvements.

5.7.1 Hop Size

The only difference between the two baseline programs used in the two sets of experiments is their hop size. It seems that smaller hop size produced better results: the baseline failed 11 times with the 16ms hop size while it failed 15 times with the 32ms hop size. All eight excerpts from Beethoven's *Sonata No. 1* were failed by the baseline using a 32ms hop size; however, only two of them were failed by the baseline using a 16ms hop size, and all

eight achieved noticeably higher accuracy with the lower hop size. As already discussed in Section 5.4, this piece was performed at a much slower tempo than the default tempo set in the program. It appears that a smaller hop size might counterbalance an incorrectly set default tempo. This finding is interesting, but any explanation would be purely speculative and beyond the current scope of this dissertation.

Among the 33 excerpts that were successfully followed by both baseline programs, the ones with lower hop size received about 1.4% higher accuracy on average, as shown in Table 5.12. I conducted a paired sample t-test on their accuracy, and as seen in Table 5.13, the p-value failed to show that the difference is statistically significant at the $\alpha = 0.1$ level. Nevertheless, there is one possible explanation for why smaller hop size might produce (slightly) more accurate results. As discussed in Section 2.4.1, the mean duration of a chord (in seconds) is calculated by

$$\mu = M \frac{\Delta}{1 - p}$$

where M is the number of micro states, Δ is the hop size, and p is the self-loop probability. Because M is constrained to be an integer, the resulting mean duration usually does not end up to be the optimal value. The maximum error is $0.5\frac{\Delta}{1-p}$, which is proportional to Δ, the hop size.

	Set 1	**Set 2**
average accuracy	65.9%	67.4%

Table 5.12: Average accuracies of 33 excerpts by both baselines, with different hop sizes.

coefficient	**value**
$\bar{d}$	-0.014
t	-1.4876
df	32
p-value	0.1466

Table 5.13: Paired t-test to compare two baselines.

5.7.2 Fast vs. Slow Excerpts

There are some extremely fast excerpts used in the experiments—there can be more than six events (notes/chords) happening within one second! The speed of an excerpt is defined as the average number of events per second, i.e.,

$$\text{Speed} = \frac{\#\text{events}}{\text{excerpt length}}$$

To clarify, the fast or slow "speed" here is different from the "tempo" defined in Chapter 4. The tempo measures the number of seconds used to play a whole note, and it emphasizes the choice of interpretation in the performance; in contrast, the speed describes how many events happen in one second, which is a function of the composition of a piece. It is a subtle difference, but one can (e.g.) choose to play a fast piece at a low tempo. For example, in Figure 5.14, if one plays both bars at a steady tempo, then the first bar is as twice as fast as the second bar.

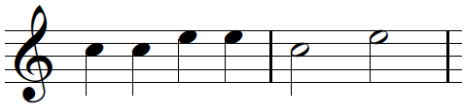


Figure 5.14: Example for comparing speed and tempo.

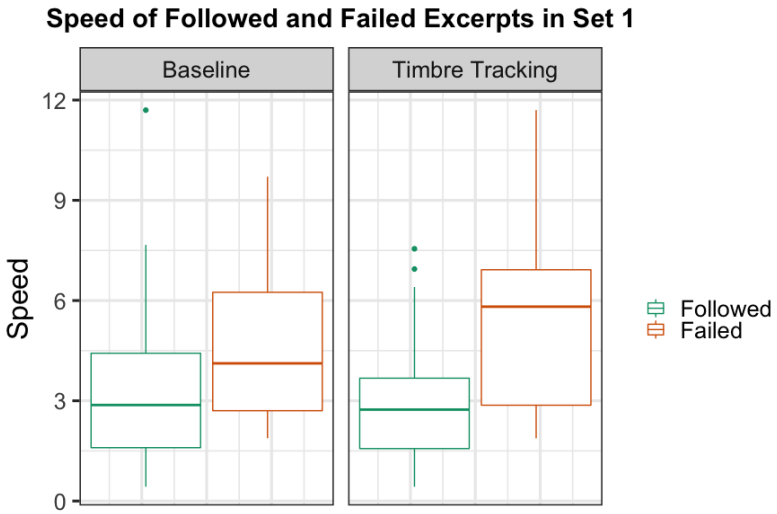


Figure 5.15: Boxplots of the speed of followed and failed excerpts in the first set of experiments.

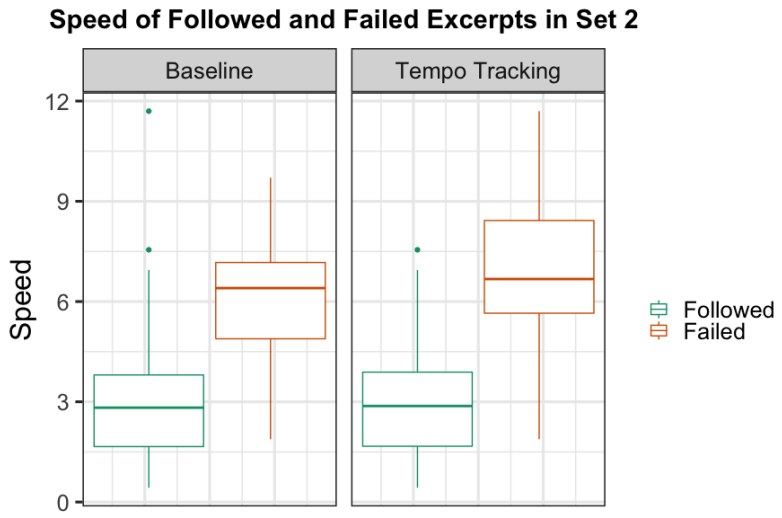


Figure 5.16: Boxplots of the speed of followed and failed excerpts in the second set of experiments.

To investigate whether the *speed* has an effect on the results, I calculated and compared the speed of the followed and failed excerpts, as shown in Figures 5.15 and 5.16. All the four pairs of box plots in Figure 5.15 and Figure 5.16 show the same trend: the failed excerpts are generally faster than the followed ones. One speculation is that there is always some delay when detecting a new chord, and this kind of delay is more troublesome for faster pieces than for slower ones. The program needs to collect a certain amount of audio evidence before it can make a confident judgement about a hypothesized chord. However, on average, a chord in a fast piece corresponds with only a handful of frames, and the program might need to move on to detecting the next chord *before* it can be confident about the current chord. When the correct hypotheses are not given enough confidence in time (in the form of probabilities), they are likely to be falsely pruned out by the beam search mechanism, which could lead to a fatal error. In contrast, a chord in slower pieces corresponds with a larger number of audio frames, and the program is provided with more opportunities to recover from a delayed detection before it has to move on to detecting the next chord. This trend is even more evident in the two new methods than in the baseline. For the timbre tracking method, one probable explanation for this pattern is that the tracked amplitudes are more accurate when there is abundant time for the Kalman filters to adapt to the actual

amplitudes of a new chord. For the tempo tracking method, as discussed in Section 5.6, fast pieces usually have greater numbers of short notes, which can cause greater approximation errors and poor estimation of the tempo. Another possible explanation is that more audio evidence of a chord improves the approximation results about the tempo, as discussed near the end of Section 4.3.3.

5.7.3 Data Model and Timing Model

Both new methods provide more flexibility than the baseline: the timbre tracking method allows the data model to adapt to the observed data, and the tempo tracking method allows the tempo to deviate from the initially set tempo value, thus making the timing model flexible. The balance between flexibility and reasonable constraints is governed by several important parameters, as mentioned in Section 5.3. These parameters can be learned and further improved by training, in ways beyond the scope of this dissertation. Nevertheless, we can speculate about the reasons why the current version of the tempo tracking algorithm showed more evident improvement than the timbre tracking algorithm.

First, there are different amounts of information loss during the approximation operations in these two models. The tempo tracking model is essentially a hidden *semi*-Markov model (HSMM), in which the transition probabilities depend on the age of a state. The timbre tracking model, in contrast, has a structure more similar to the hidden Markov model (HMM), except that every discrete state is associated with a series of partial amplitudes. There are more states in an HSMM than in an HMM, because an HSMM needs to represent states with different ages, as well as different index labels. During the approximation operations at each audio frame, the timbre tracking model loses more information than the tempo tracking model does, because the former merges nodes with the same label regardless of their ages, but the latter only merges nodes of the same label *and* age.

Second, improving the timing model might be a more urgent issue than getting a more accurate data model. As shown in Table 5.4, all eight excerpts failed by the baseline

were successfully followed after correcting the default tempo value—it strongly indicates that an accurate timing model is quite important. The goal of the score-following task is essentially about timing: the moment *when* a chord starts in a previously unheard musical performance. The baseline can end up modeling the timing information poorly because there is insufficient basis for predicting the timing of a performance beyond the recommendations from the score. In contrast, the data model in the baseline, despite being "fixed", is already of good quality, because the data model is comparatively easier to predict—although we are not sure about the amplitudes, at least we know *where* the peaks should be in a spectrum.

Third, a flexible timing model is *necessary* for following polyphonic music like piano music. The baseline program, despite using a fixed timing model, rarely fails on monophonic music. Events in monophonic music are easier to distinguish from each other: no shared notes by adjacent chords, no extended sound from previous notes because of pedaling, no blurring effects. Therefore, the data model for monophonic music can be quite discriminating: correct hypotheses receive much higher data likelihoods than incorrect ones. Even if the timing model is incorrect (with unhelpful prior probabilities), the reliable data model can still "force" the program to proceed at a lower or higher tempo than the default value. However, in piano music, the data model is generally less discriminating; when the timing model is incorrectly assumed, there is not enough "energy" from the data model to correct the tempo. In contrast, the *flexible* timing model generally holds less "resistance" toward the guidance of the data model. In fact, these two aspects work together in the tempo tracking method: the information provided by the data model, however little, improves the "merging" results in the tempo tracking method by giving higher weights to more likely nodes in the mixture, thus resulting in a better estimation of the tempo (discussed in Section 4.3.3).

Of the three explanations for the better results from the tempo tracking method versus the timbre tracking method, the first one points to their dissimilarity in technical details, while the latter two emphasize the importance of a good timing model. In summary, through

the experimental results and the above discussion, improving the timing model through tracking the tempo seems to be the more promising and necessary direction for piano score following.

CHAPTER 6

Conclusion

Following monophonic music is relatively easy because we can get quite discriminating data models to distinguish neighboring notes, so that an incorrect timing model tends not to cause any problems. In contrast, following polyphonic piano music is one of the most challenging cases in the score-following realm, mainly because neighboring chords can sound similar due to shared notes and harmonics, pedaling, or the blurring effect. The resulting unreliable data model makes a score-following program susceptible to imperfect timing models. It appears that improving the timing model is more urgent than getting a more accurate data model in piano score following, possibly because even an accurate data model may not be discriminating enough to correct the impact of flawed timing models. The tempo tracking method provides an improved timing model by allowing the tempo to adapt to the data using a switching Kalman filter. This new method produced better results than one of the state-of-the-art programs which uses a fixed default tempo throughout a piece: the tempo tracking method achieved higher average accuracy, a lower rate of failure, and a higher rate of recovery from mistakes in the experiments on 50 excerpts of real piano performance recordings. The other new method, the timbre tracking method, although it did not show improvements in terms of average accuracy, did seem to improve the data model and led to higher accuracy in later frames.

The two new methods represent the first time a switching Kalman filter has been used for the score-following problem. These mathematical models are interesting in their own right. In the experiments, the tempo tracking method achieved 4.1% higher average accuracy than the state-of-the-art baseline program on successfully followed excerpts, more if including failed excerpts. This brings us one step closer to many musical applications that depend on piano score following. In addition, the tempo tracking method is generalizable and

can be applied to other instruments. The tracked tempo information is also meaningful for automatic accompaniment systems, and large-scale musical performance analysis. The timbre tracking method, meanwhile presents a general framework for tracking the changing features within a chord—a new domain that has not been explored before. This new method provides a foundation for future exploration in this domain in greater depth.

Another contribution of this work is the new evaluation method: frame-wise accuracy. This method is simple and intuitive, and most importantly, it avoids the trade-off issue between accuracy and promptness that has challenged all previous evaluation methods. Using this evaluation method, the experiments on real piano performance recordings provide valuable insights for future improvements, especially through the individual case studies of particular excerpts.

While the tempo tracking method helped raise the average frame-wise accuracy from 65% to 69.1% on successfully followed excerpts, it still failed on nine excerpts. Three factors seem to limit this method. First, this approach does not directly address the challenges brought by pedaling, repeated notes, and the blurring effect in piano music, and the program can still suffer from high uncertainty or delayed detection in those situations. Second, the program is more likely to fail on fast pieces; fast playing often results in blurring, and moreover involves shorter notes, for which estimation of the tempo can be less accurate, perhaps due to larger approximation errors for onsets. Third, in places where the tempo changes suddenly in a performance because of the performer's musical interpretations, the program might falsely treat the change as an outlier, because it is not modeled with high-level musical knowledge that can help evaluate musical interpretations. Out of the three, addressing the first factor seems to be most valuable because it is a universal problem in all approaches to piano score following. Future researchers may consider modeling pedaling, or including a new feature for detecting a new starting chord—for example, if the spectrum difference between neighboring frames is always positive at some frequencies, it indicates that a new chord is starting.

Despite offering an interesting new approach, the timbre tracking method did not show better results than the baseline. However, I should point out that the current version of the timbre tracking algorithm is just a starting point, and has considerable potential for improvement beyond the scope of these experiments. For example, the parameters were set manually, whereas training them on a larger data set using machine learning techniques could potentially lead to better results. One important parameter among these is the decay rate: the current version has the same decay rate for all frequencies, but in reality, higher frequencies usually decay faster than lower ones. Perhaps learning the actual rates for different frequencies from the real data would bring better results.

It takes years of musical training for a musician to be able to follow a score as complex as the ones used here. We should be impressed by these model-based score-following machines, as they do not involve any sort of training that's comparably intense. Humans and machines do not always agree on the challenging aspects when following a piece: measures that seem to be easy for humans can be really challenging for machines, and vice-versa. But only through discerning the unique difficulties faced by machines, can we continue making them smarter.

Bibliography

Agrawal, R. and Dixon, S. (2019). A hybrid approach to audio-to-score alignment. In *International Conference on Machine Learning*. (Cited on page 9.)

Arzt, A. and Widmer, G. (2010). Simple tempo models for real-time music tracking. In *Proceedings of the Sound and Music Computing Conference (SMC)*. (Cited on page 9.)

Arzt, A., Widmer, G., and Dixon, S. (2008). Automatic page turning for musicians via real-time machine listening. In *ECAI*, pages 241–245. (Cited on page 4.)

Benetos, E., Dixon, S., Giannoulis, D., Kirchhoff, H., and Klapuri, A. (2013). Automatic music transcription: challenges and future directions. *Journal of Intelligent Information Systems*, 41(3):407–434. (Cited on page 1.)

Cont, A. (2006). Realtime audio to score alignment for polyphonic music instruments, using sparse non-negative constraints and hierarchical HMMs. In *2006 IEEE International Conference on Acoustics Speech and Signal Processing Proceedings*, volume 5, pages V–V. IEEE. (Cited on page 7.)

Cont, A. (2008). Antescofo: Anticipatory synchronization and control of interactive parameters in computer music. In *International Computer Music Conference (ICMC)*, pages 33–40. (Cited on page 9.)

Cont, A. (2010). A coupled duration-focused architecture for real-time music-to-score alignment. *IEEE transactions on pattern analysis and machine intelligence*, 32(6):974–987. (Cited on pages 6, 8, 9, and 30.)

Cont, A. (2011). On the creative use of score following and its impact on research. In *SMC 2011: 8th Sound and Music Computing conference*. (Cited on pages 5 and 9.)

Cont, A., Schwarz, D., Schnell, N., and Raphael, C. (2007). Evaluation of real-time audio-to-score alignment. In *International Symposium on Music Information Retrieval (ISMIR)*. (Cited on pages 11 and 73.)

Cuvillier, P. (2016). *On temporal coherency of probabilistic models for audio-to-score alignment*. PhD thesis, Université Pierre et Marie Curie-Paris VI. (Cited on pages 9 and 30.)

Dannenberg, R. B. (1984). An on-line algorithm for real-time accompaniment. In *ICMC*, volume 84, pages 193–198. (Cited on page 6.)

Dannenberg, R. B. and Hu, N. (2003). Polyphonic audio matching for score following and intelligent audio editors. In *ICMC*, pages 27–34. (Cited on page 8.)

Dannenberg, R. B. and Raphael, C. (2006). Music score alignment and computer accompaniment. *Communications of the ACM*, 49(8):38–43. (Cited on pages 1 and 4.)

Dixon, S. (2005). Live tracking of musical performances using on-line time warping. In *Proceedings of the 8th International Conference on Digital Audio Effects*, pages 92–97. Citeseer. (Cited on page 8.)

Duan, Z. and Pardo, B. (2011). A state space model for online polyphonic audio-score alignment. In *2011 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 197–200. IEEE. (Cited on page 9.)

Durbin, J. and Koopman, S. J. (2012). *Time series analysis by state space methods*. Oxford university press. (Cited on page 34.)

Ensemble, S. and Suurmond, P. (2001). Comparser. www.schreck.nl/eng/comp.html. (Cited on page 5.)

Fechner, G. T., Howes, D. H., and Boring, E. G. (1966). *Elements of psychophysics*, volume 1. Holt, Rinehart and Winston New York. (Cited on page 76.)

Fernández, J. D. and Vico, F. (2013). AI methods in algorithmic composition: A comprehensive survey. *Journal of Artificial Intelligence Research*, 48:513–582. (Cited on page 1.)

Ghahramani, Z. (1997). Learning dynamic Bayesian networks. In *International School on Neural Networks, Initiated by IIASS and EMFCSC*, pages 168–197. Springer. (Cited on page 25.)

Ghahramani, Z. and Hinton, G. E. (2000). Variational learning for switching state-space models. *Neural computation*, 12(4):831–864. (Cited on page 39.)

Gröchenig, K. (2001). *Foundations of time-frequency analysis,*. Springer Science & Business Media. (Cited on page 21.)

Grubb, L. and Dannenberg, R. B. (1997). A stochastic method of tracking a vocal performer. In *ICMC*. (Cited on page 6.)

Grubb, L. V. (1998). *A probabilistic method for tracking a vocalist*. PhD thesis, Carnegie-Mellon University. (Cited on page 6.)

Hawthorne, C., Stasyuk, A., Roberts, A., Simon, I., Huang, C.-Z. A., Dieleman, S., Elsen, E., Engel, J., and Eck, D. (2018). Enabling factorized piano music modeling and generation with the MAESTRO dataset. (Cited on page 71.)

Hook, J. (2011). How to perform impossible rhythms. *Music Theory Online*, 17(4):1–14. (Cited on page 92.)

İzmirli, Ö. and Dannenberg, R. B. (2010). Understanding features and distance functions for music sequence alignment. In *ISMIR*, pages 411–416. Citeseer. (Cited on page 9.)

Jiang, Y. and Raphael, C. (2019). Piano score-following by tracking note evolution. (Cited on pages 3, 18, and 73.)

Jiang, Y., Ryan, F., Cartledge, D., and Raphael, C. (2019). Offline score alignment for realistic music practice. (Cited on page 10.)

Joder, C., Essid, S., and Richard, G. (2010). A conditional random field viewpoint of symbolic audio-to-score matching. In *Proceedings of the 18th ACM international conference on Multimedia*, pages 871–874. ACM. (Cited on page 9.)

Kalman, R. E. (1960). A new approach to linear filtering and prediction problems. (Cited on page 16.)

Kamenetsky, S. B., Hill, D. S., and Trehub, S. E. (1997). Effect of tempo and dynamics on the perception of emotion in music. *Psychology of Music*, 25(2):149–160. (Cited on page 71.)

Kim, C.-J. (1994). Dynamic linear models with Markov-switching. *Journal of Econometrics*, 60(1-2):1–22. (Cited on page 39.)

Montecchio, N. and Cont, A. (2011). A unified approach to real time audio-to-score and audio-to-audio alignment using sequential Montecarlo inference techniques. In *ICASSP 2011: Proceedings of International Conference on Acoustics, Speech and Signal Processing*, pages 193–196. IEEE. (Cited on page 9.)

Müller, M., Kurth, F., and Röder, T. (2004). Towards an efficient algorithm for automatic score-to-audio synchronization. In *ISMIR*. (Cited on page 7.)

Müller, M., Mattes, H., and Kurth, F. (2006). An efficient multiscale approach to audio synchronization. In *ISMIR*, volume 546, pages 192–197. Citeseer. (Cited on page 7.)

Murphy, K. P. (1998). Switching Kalman filters. (Cited on pages 16, 35, 37, and 39.)

Murphy, K. P. and Russell, S. (2002). Dynamic Bayesian networks: representation, inference and learning. (Cited on page 25.)

Nakamura, T., Nakamura, E., and Sagayama, S. (2013). Acoustic score following to musical performance with errors and arbitrary repeats and skips for automatic accompaniment. *Proceedings of Sound and Music Computing*, pages 299–304. (Cited on page 10.)

Nakamura, T., Nakamura, E., and Sagayama, S. (2016). Real-time audio-to-score alignment of music performances containing errors and arbitrary repeats and skips. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, 24(2):329–339. (Cited on page 10.)

Orio, N. and Déchelle, F. (2000). Score following using spectral analysis and hidden Markov models. In *ICMC: International Computer Music Conference*, pages 1–1. (Cited on page 7.)

Orio, N., Lemouton, S., and Schwarz, D. (2003). Score following: State of the art and new developments. (Cited on page 6.)

Orio, N. and Schwarz, D. (2001). Alignment of monophonic and polyphonic music to a score. In *International Computer Music Conference (ICMC)*, pages 1–1. (Cited on page 7.)

Otsuka, T., Nakadai, K., Takahashi, T., Ogata, T., and Okuno, H. (2011). Real-time audio-to-score alignment using particle filter for coplayer music robots. *EURASIP Journal on Advances in Signal Processing*, 2011(1):384651. (Cited on page 9.)

Pardo, B. and Birmingham, W. (2005). Modeling form for on-line following of musical performances. In *Proceedings of the National Conference on Artificial Intelligence*, volume 20, page 1018. Menlo Park, CA; Cambridge, MA; London; AAAI Press; MIT Press; 1999. (Cited on page 10.)

Peeters, G. and Papadopoulos, H. (2011). Simultaneous beat and downbeat-tracking using a probabilistic framework: Theory and large-scale evaluation. *IEEE Transactions on Audio, Speech, and Language Processing*, 19(6):1754–1769. (Cited on page 1.)

Puckette, M. and Lippe, C. (1992). Score following in practice. In *Proceedings of the International Computer Music Conference*, pages 182–182. International Computer Music Association. (Cited on page 6.)

Rabiner, L. R. (1989). A tutorial on hidden Markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 77(2):257–286. (Cited on page 26.)

Raphael, C. (1999). Automatic segmentation of acoustic musical signals using hidden Markov models. *IEEE transactions on pattern analysis and machine intelligence*, 21(4):360–370. (Cited on pages 7, 27, and 30.)

Raphael, C. (2004). A hybrid graphical model for aligning polyphonic audio with musical scores. In *ISMIR*, pages 387–394. Citeseer. (Cited on pages 8 and 56.)

Raphael, C. (2006). Aligning music audio with symbolic scores using a hybrid graphical model. *Machine learning*, 65(2-3):389–409. (Cited on pages 19 and 30.)

Raphael, C. (2010). Music Plus One and machine learning. In *ICML*, pages 21–28. (Cited on page 7.)

Repp, B. H. (1990). Patterns of expressive timing in performances of a Beethoven minuet by nineteen famous pianists. *The Journal of the Acoustical Society of America*, 88(2):622–641. (Cited on page 5.)

Rothstein, J. (1995). *MIDI: A comprehensive introduction*, volume 7. AR Editions, Inc. (Cited on page 2.)

Runnalls, A. R. (2007). Kullback-Leibler approach to Gaussian mixture reduction. *IEEE Transactions on Aerospace and Electronic Systems*, 43(3):989–999. (Cited on page 37.)

Schoonderwaldt, E., Askenfelt, A., and Hansen, K. F. (2005). Design and implementation of automatic evaluation of recorder performance in IMUTUS. In *In Proceedings of the International Computer Music Conference (ICMC)*. (Cited on page 5.)

Soulez, F., Rodet, X., and Schwarz, D. (2003). Improving polyphonic and poly-instrumental music to score alignment. (Cited on page 7.)

Vercoe, B. (1984). The synthetic performer in the context of live performance. In *Proceedings of International Computer Music Conference*, pages 199–200. (Cited on page 6.)

Welch, G., Bishop, G., et al. (1995). An introduction to the Kalman filter. (Cited on page 34.)

Winston, P. H. and Brown, R. H. (1979). Artificial intelligence: an MIT perspective. *Cambridge, Mass*. (Cited on page 1.)

Curriculum Vitae

EDUCATION

Indiana University, Bloomington, Indiana USA
Doctor of Philosophy in Computer Science *August 2020*
Master of Science in Computer Science *March 2016*
Beijing University of Posts and Telecommunications, Beijing China
Bachelor of Engineering in Electrical Engineering *June 2012*

PUBLICATIONS

Jiang, Y., & Raphael, C. (2020, accepted). Score Following with Hidden Tempo Using a Switching State-space Model. In International Society for Music Information Retrieval (ISMIR).

Jiang, Y., & Raphael, C. (2019). Piano Score-following by Tracking Note Evolution. In Proc. Sound and Music Computing (SMC).

Jiang, Y., Ryan, F., Cartledge, D., & Raphael, C. (2019). Offline Score Alignment for Realistic Music Practice. In Proc. Sound and Music Computing (SMC).

Jiang, Y., & Raphael, C. (2015). Instrument Identification in Optical Music Recognition. In International Society for Music Information Retrieval (ISMIR) (pp. 612-617).

TEACHING

Instructor *(independently designed and proposed course)*
CS-A290/590 Topic: Programming with R Su19
Associate Instructor
CS-B365 Data Analysis and Mining Fa18, Sp19, Fa19, Sp20
CS-B536 Advanced Operating Systems Fa14
CS-A321 Computing Tools for Scientific Research Sp13
CS-A201 Introduction to Programming I (Python) Fa12, Fa13, Sp14, Sp15, Fa15

INDUSTRY EXPERIENCE

Google Inc., Los Angeles, California USA *May-August 2017*
Software Engineering Intern Team: Ads Exclusions
Google Inc., Mountain View, California USA *May-July 2016*
Software Engineering/Network Engineering Intern Team: Corporate Networking