Realtime Tempo Tracking using Kalman Filtering

Tim van Kasteren

Supervisors:
Dr. A. Taylan Cemgil
Dr. ir. B.J.A. Kröse

March 2006

University of Amsterdam

Tim van Kasteren

Realtime Tempo Tracking using Kalman Filtering
Master's Thesis in Artificial Intelligence

Supervisors:
Dr. A. Taylan Cemgil
Dr. ir. B.J.A. Kröse

March 30, 2006

University of Amsterdam
Department of Computer Science
Intelligent Autonomous Systems Group
Kruislaan 403
1098 SJ Amsterdam
The Netherlands

Abstract

In this thesis a probabilistic model for timing deviations in expressive music performances is used to perform tempo tracking in realtime. A switching Kalman filter calculates the optimal estimate of the tempo using the prediction from the model and the measurement from a realtime performance. Using a particle filter various hypotheses are tested to determine what interpretation of the score fits a given performance best. This allows us to perform tempo tracking on a musical performance of which the score is not known in advance. The realtime implementation of this approach is discussed and a number of tests is performed to determine whether this approach qualifies for realtime processing.

Contents

Chapter 1

Introduction

One of the most challenging topics in the field of computer music is that of interactive music performance systems. These systems are able to 'listen' to the actions of a musical performer and generate responses in realtime. However, because of the diversity of the domain, for example: different genres, polyphony, fluctuating tempo, this task has proved to be rather difficult and has been split up in a number of subtasks. In this thesis we will explore one of these subtasks known as tempo tracking, a challenging task in itself with a number of very interesting applications.

In tempo tracking we try to determine the tempo of a piece of music. As this tempo is not likely to be constant throughout a performance, the program follows or tracks the tempo as the music progresses. Tempo refers to the speed at which a musical performance is played and is often expressed as the number of beats per minute. This number corresponds to the rate at which most people would tap or clap in time with the music. Another way to express the tempo is as the time interval between beats, referred to as the inter-beat interval or period. With the notion of beats in mind, tempo tracking can be seen as an inverted metronome. A metronome is used to provide a performer with a constant tempo by playing clicks at the time of the beat. In tempo tracking this process is reversed, the performer plays his piece and the tempo tracking software listens to the performance in order to generate the clicks at the matching tempo.

In this introduction we will first go over some basic music terminology in section 1.1 to provide us with a language for discussing various musical concepts. To get a good understanding of how tempo tracking works, an overview of related work will be explored in section 1.2, we will look into various features that can be used for tempo tracking and several approaches from other researchers. We will then go over possible applications in section 1.3, learning how the results of tempo tracking can be used in interesting applications. In section 1.4 we will formulate the problem to be discussed in this thesis and in section 1.5 we conclude the chapter with an outline of the remainder of this thesis.

1.1 Music Terminology

Musicians use sheet music in order to read what they have to play. This sheet music consists of notes and all sorts of other symbols expressing what actions the musician is to perform. In more general terms we speak of a score, which is the sheet music showing all the instruments of

a given performance. Although a large amount of actions can be expressed using the symbols in sheet music, for the case of tempo tracking we are mainly interested in notes and rests. A note expresses how long and at what pitch the performer should use his instrument, while a rest simply indicates the musician should not do anything for a certain amount of time. An example of a piece of score is given in figure 1.1, above this example little arrows indicate the position of the beat. The beat is a basic time unit used by musicians to keep track of their position in the music. As this example shows, not every note coincides with a beat and not every beat coincides with a note either. We can therefore distinguish between the moment in time a beat is played and the moment in time a note is played. The starting time of a note is referred to as the onset time, while the time the note stops is called the offset time. Using this terminology, we define the time between the onset time and the offset time as the duration and the time between the onset times of two consecutive notes as the inter-onset interval.

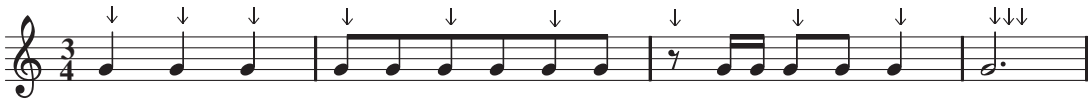

Figure 1.1: An example of a simple score, the arrows indicate the beats.

Music can either be monophonic or polyphonic, meaning either one note is played at a time or several notes are played together. In either case, the music will have to be captured as data input to allow a computer program to process it. One approach is to record the music using a microphone, resulting in a raw audio signal. Because this signal in itself contains no abstract information about the music, some preprocessing will need to be performed to extract such information. Another approach is to use MIDI which stands for Musical Instrument Digital Interface and is a standard communication protocol for instruments and computers. MIDI does not transmit the audio signal itself, but instead transmits digital information about a musical performance. As this MIDI data contains a lot of symbolic information, it is ideal for use in the field of computer music.

1.2 Related Work

With the basic terminology of music explained we can look at various approaches of performing tempo tracking. In this section we will first look at what features we can extract from MIDI data and raw audio signal for the use of tempo tracking. We will then discuss a number of methods used by other researchers to perform tempo tracking. [Gouyon2005]

1.2.1 Feature Extraction

The first step in developing software capable of tempo tracking is creating a feature list. In other words, we need to decide on how to parse or filter the incoming data into a sequence of features that will give us clues about the tempo. A wide range of features can be used for performing tempo tracking. One of the most common features is the onset time, which is the moment in time a note is played, indicating the start of a note. The extraction of onset times

from MIDI data is very easy because of the availability of symbolic data. More complex is the automatic extraction from audio signals, especially in the case of polyphonic music. An overview of musical onset detection can be found in Bello(2003). In addition to onset times, some systems also handle the duration of notes, like onset times durations can be easily parsed from MIDI data, but cannot be computed reliably from audio data. Experiments done by Snyder show that onsets and durations alone are sufficient for the perception of a beat in ragtime music [Snyder2001].

Although the onset time and duration of a note provide enough information to perform tempo tracking, other features can be included to achieve better performance. One such feature is the relative amplitude which can be easily extracted from MIDI or audio data and is used by some systems for weighting onsets (see [Smith1996], [Gasser1999] and [Dixon2001]). Although the tempo in music is associated mainly with rhythmic features, Dixon uses pitch to calculate the 'salience' of musical events and is shown to improve the performance of their system [Dixon2000]. Pitch can be easily obtained from MIDI data, but cannot be reliably extracted from polyphonic audio. Finally, chords can be used as a feature in two ways: by counting the number of simultaneous notes and using this as a measure of accentuation [Rosenthal1992] or by detecting harmonic change as evidence of the start of a new measure (see [Goto1999] and [Temperley1999]). Just like pitch, chords are easily readable in MIDI data, but are much harder to derive from audio data.

There are some systems that do not focus on note onsets at all; these systems use a lower abstraction level referred to as frames. A frame is a short chunk of audio (typically about 10 milliseconds), from which both time and frequency domain features can be computed. The simplest feature that can be extracted from these frames is the spectral energy, which can be calculated either for the entire frame or for frequency sub-bands of the frame. Using the assumption that low-frequency instruments provide much of the rhythmic information, Alghoniemy focusses on energy in low-frequency components [Alghoniemy1999]. Others decompose the signal into several sub-bands, for each of these sub-bands the energy is computed and they are added back together (see [Vercoe1997] and [Tzanetakis2002]). Instead of focussing on frame energy values, some systems measure the variation of energy between consecutive frames, like Foote where the cosine distance between the magnitude spectra of consecutive frames is used [Foote2001].

Each of these features can be used to provide a program with information about the tempo of a performance. Whatever feature set is used, the actual power of a method lies in using the features appropriately to determine the tempo.

1.2.2 Methods

A lot of different approaches have been used in the design of tempo trackers, still each of these approaches can be described by a set of state variables and a set of actions manipulating this state. The state variables are used to represent the inner state of the tracker and usually include the tempo and some measure expressing the position of the current beat, the actions dictate how this state should be altered when a set of features is observed.

In one of the most simple approaches known as adaptive oscillators, the next beat position is predicted by taking the current beat position and adding the period (the time between two consecutive beats). The closest observed feature to this prediction is then used to update the state variables accordingly. This approach can be expanded by connecting several oscillators in a network allowing them to interact in order to model several metrical levels jointly (see

[Large1994] and [McAuley1995]).

Rule-based approaches use the period together with the first and current beat as the state variables. A set of 'if-then' rules adapts these variables as each feature is observed and predicts the next beat (see [Longuet-Higgins1982] and [Desain1999]).

An agent based approach is used by Dixon, the state of an agent consists of the period and the position of the beat. Furthermore, the agent maintains a history of beat times. Several agents are used making the task a multiple-hypothesis search approach. Each agent does its prediction and receives a score indicating the quality of its prediction, the agent with the highest score is chosen [Dixon2001].

Finally Cemgil models tempo tracking using a dynamical system. The system uses the period and time of the beat as hidden state variables, the transition from one state to another is modeled by a set of state equations. To account for possible tempo variations a Gaussian noise term is added to the model. The hidden state variables are estimated by combining the observed features with the dynamical system using a Kalman filter [Cemgil2004].

1.3 Possible Applications

At the start of this chapter we noted that tempo tracking is a subtask of interactive music performance systems. However, when performing tempo tracking by itself it can also be used for generating some simple interactive music performances. With the tempo information available in realtime it is possible to play a drum loop at the same speed as the performing artist, providing him with a percussion background. Because tempo tracking does not take things like pitch into account, it is not possible to generate matching chords automatically. However, some musical genres such as the blues use a very simple and repetitive pattern for chords, therefore it is possible to use such a pattern in a loop and simply tell the musical performer to play the blues.

Tempo tracking can also be used to create score following systems. These systems are able to listen to a performer and indicate at which position in the score the artist is playing. This can be used to guide the artist through the score in a sort of karaoke manner, making sure he knows which note to read at all times. Score following can also be used for teaching purposes, as the application knows what is supposed to be played, it can work as a tutor correcting the artist when not playing the precise notes the score dictates.

Finally, tempo tracking is also a subtask of a field known as automatic music transcription. Here the goal is to automatically generate sheet music based on a given musical performance without knowing anything about the performance in advance. This is a very challenging task as musical performances allow various interpretations and even professional musicians do not always agree with each other on such interpretations.

1.4 Problem Description

With the ultimate goal being the creation of an interactive music performance system, most of the methods for tempo tracking discussed so far do not qualify to our needs. We need an algorithm which can be used in realtime processing, providing us with accurate results and being able to recover from errors in the past. All this has to be done in a reasonably short timespan for the program to be able to respond to the incoming data.

Cemgil uses a probabilistic generative model for timing deviations in expressive music performances. A Kalman filter uses this model in combination with onset times to form an estimation of the tempo. To determine the most probable interpretation, several methods are applied and compared to determine which performs best. Although these comparisons were made based on offline testing scenarios, the methods do allow an online implementation. In performing tempo tracking the model also takes the rhythmic structure of the music into account, it is able to determine the inter-onset intervals therefore providing us with more information than just the tempo of a performance. Information about the rhythmic structure can proof useful when the results of tempo tracking are combined with other tasks to form an interactive music performance system [Cemgil2004].

In tests performed in the offline case the model performed best when using a particle filter for processing. This method is capable of recovering from errors and provide us with accurate results, the question is whether it will be fast enough for realtime processing.

1.5 Thesis Outline

In the following chapters we will explain the model taken from Cemgil piece by piece and will determine if it qualifies for performing tempo tracking in realtime [Cemgil2004].

In chapter 2 we will discuss some basic mathematical principles that we will use throughout this thesis. We will discuss probability theory, linear dynamical systems and the Kalman filter and will elaborate on each of these by working out an example.

Chapter 3 introduces a mathematical model which allows us to perform calculations on musical data. A number of variables are introduced and their relationship will be explained, all this leads to a linear dynamical system model for the tempo in a musical performance.

Methods for performing tempo tracking are discussed in chapter 4. We start with a simple case in which the score of a musical performance is known in advance. Once the difficulties in this scenario have been explained we discuss how to perform tempo tracking when we are dealing with an unknown score.

In chapter 5 we will discuss the architecture of the program created to perform tempo tracking. Each module of the program will be discussed and some implementation choices will be explained.

Chapter 6 reviews the results from several tests performed on the created program. To determine if it is qualified for performing tempo tracking in realtime, the accuracy of the implementation and the speed of the calculations are tested.

Finally we discuss the conclusions of all this in chapter 7 looking at future work and improvements of the model.

Chapter 2

Basic theory

In this chapter we will discuss some basic mathematical principles that will be used to solve the tempo tracking task. We will discuss these principles in their general form and then illustrate them with a simple example. The application of these principles to the tempo tracking case is discussed in later chapters.

In section 2.1 we discuss some basic probability theory that we will use in our model. Section 2.2 explains linear dynamical systems, as music is not static but progresses in time we explain how linear dynamical systems provide a way to model the change of a system over time. Finally in section 2.3 we discuss the Kalman filter, which provides us with an optimal recursive method for filtering out noise.

2.1 Probability Theory

This section lays the foundation for a model of stochastic events and processes which deterministic models cannot adequately describe. As no system is driven only by its own inputs, but also by disturbances which we can neither control nor model deterministically, probability theory provides a way to sufficiently include such disturbances. Also because sensors do not provide perfect and complete data about a system we need probability theory to take these shortcomings into account [Maybeck1979]. The topics in this section are taken from an introductory text book on probability by Dimitri Bertsekas [Bertsekas2002].

2.1.1 Principles of Probability Theory

A probabilistic model has two main ingredients.

- **The sample space** Ω, which is the set of all possible outcomes of an experiment.
- **The probability law** which assigns to a set A of possible outcomes (also called an event) a nonnegative number $P(A)$ (called the probability of A) that encodes our knowledge or belief about the collective 'likelihood' of the elements of A.

The probability law has to satisfy the following axioms.

- **(Nonnegativity)** $P(A) \geq 0$, for every event A.

- **(Additivity)** If A and B are two disjoint events, then the probability of their union satisfies

$$P(A \cup B) = P(A) + P(B)$$

- **(Normalization)** The probability of the entire sample space Ω is equal to 1, that is, $P(\Omega) = 1$.

2.1.2 Conditional Probability

Another important concept is conditional probability, which allows us to reason about the outcome of an experiment based on partial information. We speak of the conditional probability of A given B, denoted by $P(A \mid B)$. The conditional probability of an event A, given an event B with $P(B) > 0$, is defined by

$$P(A \mid B) = \frac{P(A \cap B)}{P(B)} \tag{2.1}$$

Conditional probabilities can also be viewed as a probability law on a new universe B. So B can be viewed as a subuniverse of the entire sample space Ω. Now $P(A \mid B)$ gives the probability of A from the perspective of the subuniverse B.

2.1.3 Total Probability Theorem

The Total Probability Theorem states that if we have $\omega_1, ..., \omega_n$ disjoint events that form a partition of the entire sample space and assume that $P(\omega_i) > 0$, for all i. Then, for any event x, we have

$$\begin{aligned} P(x) &= P(\omega_1 \cap x) + ... + P(\omega_n \cap x) \\ &= P(\omega_1)P(x \mid \omega_1) + ... + P(\omega_n)P(x \mid \omega_n). \end{aligned} \tag{2.2}$$

As the formula shows, the probability for x is equivalent to the sum events ω_i intersecting with x. This is quite obvious as the union of the disjoint events ω_i is $\cup_{i=1}^{n} \omega_i = \Omega$ and the intersection between $x \cap \Omega = x$.

2.1.4 Bayes' Rule

An important theorem in probability theory is Bayes' rule, it relates conditional probabilities of the form $P(\omega \mid x)$ with conditional probabilities of the form $P(x \mid \omega)$, in which the order of the conditioning is reversed. We take $\omega_1, ..., \omega_n$ to be disjoint events that form a partition of the sample space, and assume that $P(\omega_i) > 0$, for all i. Then, for any event x such that $P(x) > 0$, we have

$$P(\omega_i \mid x) = \frac{P(\omega_i)P(x \mid \omega_i)}{P(x)} \tag{2.3a}$$

$$= \frac{P(\omega_i)P(x \mid \omega_i)}{P(\omega_1)P(x \mid \omega_1) + ... + P(\omega_n)P(x \mid \omega_n)} \tag{2.3b}$$

Bayes' rule is often used for inference. Say we have a task that requires us to classify objects into a number of classes, when we observe an object, or rather a data point, we wish to infer which class it belongs to. The events $\omega_1, ..., \omega_n$ are various classes and x is a data point. We call $P(\omega_i)$ the *prior probability* as it reflects the background knowledge of ω_i before or prior to any observation, the term $P(x \mid \omega_i)$ reflects the *likelihood* of the data point x given class ω_i, while $P(\omega_i \mid x)$ is known as the *posterior probability* as it represents the probability after the data point has been observed. In the inference task we wish to find the most probable class ω_i given data point x, in other words we wish to find the *maximum a posterior* (MAP) probability [Mitchell1997].

2.1.5 Random Variables

So far we have discussed the probability of events, we can formulate collections of events of interest and assign a probability to every one of them. Theoretically we could be finished. However, we can deal with numerical representations of sets in a space more readily than with the abstract subsets themselves. Therefore, for quantitative analysis, we need a mapping from the sample space Ω to the real numbers. This is the reason we introduce the concept of a random variable [Maybeck1979]. A random variable is a function that maps all points in the sample space to real numbers. Most useful to our case is the use of continuous random variables, for example the mapping of time to real numbers. For continuous random variables the probability of any single event A is in fact 0, that is $p(A) = 0$. Instead we can only evaluate probability of events within some interval. A function for representing the probability of random variables is known as the *cumulative distribution function*:

$$F_X(x) = p(-\infty, x] \tag{2.4}$$

This function represents the cumulative probability of the continuous random variable X for all (uncountable) events up to and including x. Important properties of the cumulative density function are

1. $F_X(x) \to 0$ as $x \to -\infty$
2. $F_X(x) \to 1$ as $x \to +\infty$
3. $F_X(x)$ is a non-decreasing function of x.

Even more commonly used than the cumulative distribution function is its derivative, known as the *probability density function*:

$$f_X(x) = \frac{d}{dx} F_X(x) \tag{2.5}$$

This function has the following properties:

1. $f_X(x)$ is a non-negative function
2. $\int_{-\infty}^{\infty} f_X(x)dx = 1$

Finally the probability over any interval $[a, b]$ is defined as

$$p_X[a, b] = \int_a^b f_X(x)dx \tag{2.6}$$

This section was taken from Welch(2001).

2.1.6 Gaussian Random Variables

A particular random variable used often in modelling natural phenomena is known as the Normal or Gaussian random variable. The random variable X is called Normal or Gaussian if it can be described through a probability density function of the following form [Jordan2005].

$$f_X(x) = \frac{1}{(2\pi)^{n/2} \mid \Sigma \mid^{1/2}} \exp\left\{-\frac{1}{2}(x-\mu)^T\Sigma^{-1}(x-\mu)\right\} \tag{2.7}$$

Here μ represents the mean and Σ the covariance of the random variable. The normal density function is completely specified by its mean and covariance, therefore a shorthand notation is often used. The form $X \sim N(\mu, \Sigma)$ means X is a normal random variable with mean μ and covariance Σ [Brown1992]. A plot of the Gaussian density function is shown in figure 2.1.

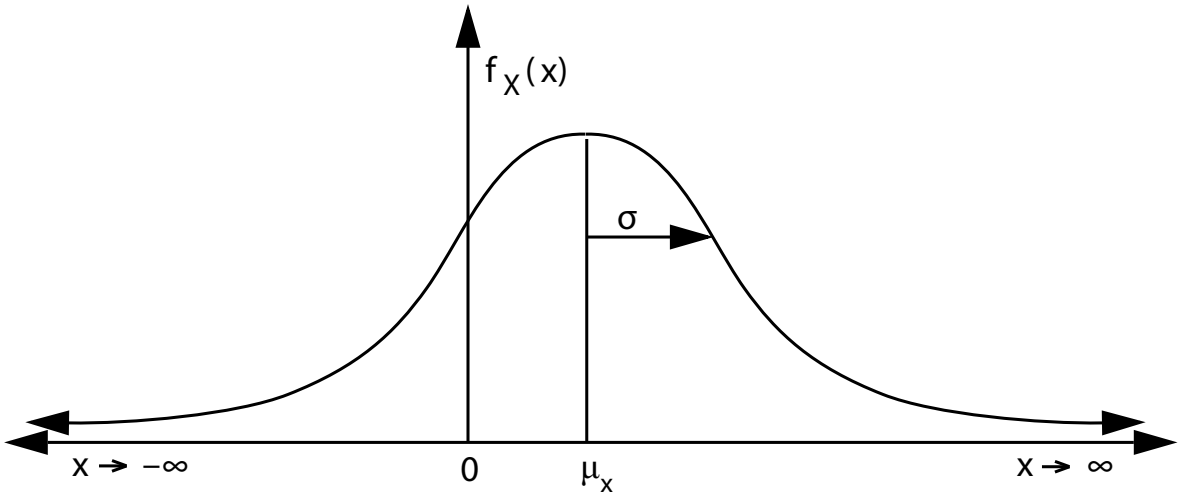

Figure 2.1: The Normal or Gaussian density function. (Taken from [Welch2001])

2.1.7 Example

To illustrate the topics discussed in the previous sections we will go over an example. This story like example is meant as an intuitive introduction to the concepts discussed, where neither messy data nor a complicated scenario distracts us from the mathematical idea. Imagine we are standing in front of a small pond, the pond contains two types of fish that differ both in size and color. One of the type of fish is known as little fish, while the other type is called big fish. Quite a bit of research has been done on the fish in this pond and for both fish types we know the mean and covariance of their size and color. We can therefore use a Gaussian random variable X to map these two features to a Euclidean space as shown in figure 2.2. We also know that there are in total 30 small fishes and a total of 10 big fishes in the pond.

What we would like to do is catch a fish and determine whether it is a little or a big fish. We can do this by first determining the size and color of the fish we caught and storing its values in a datapoint x. We can then calculate for which fish type this datapoint achieves the

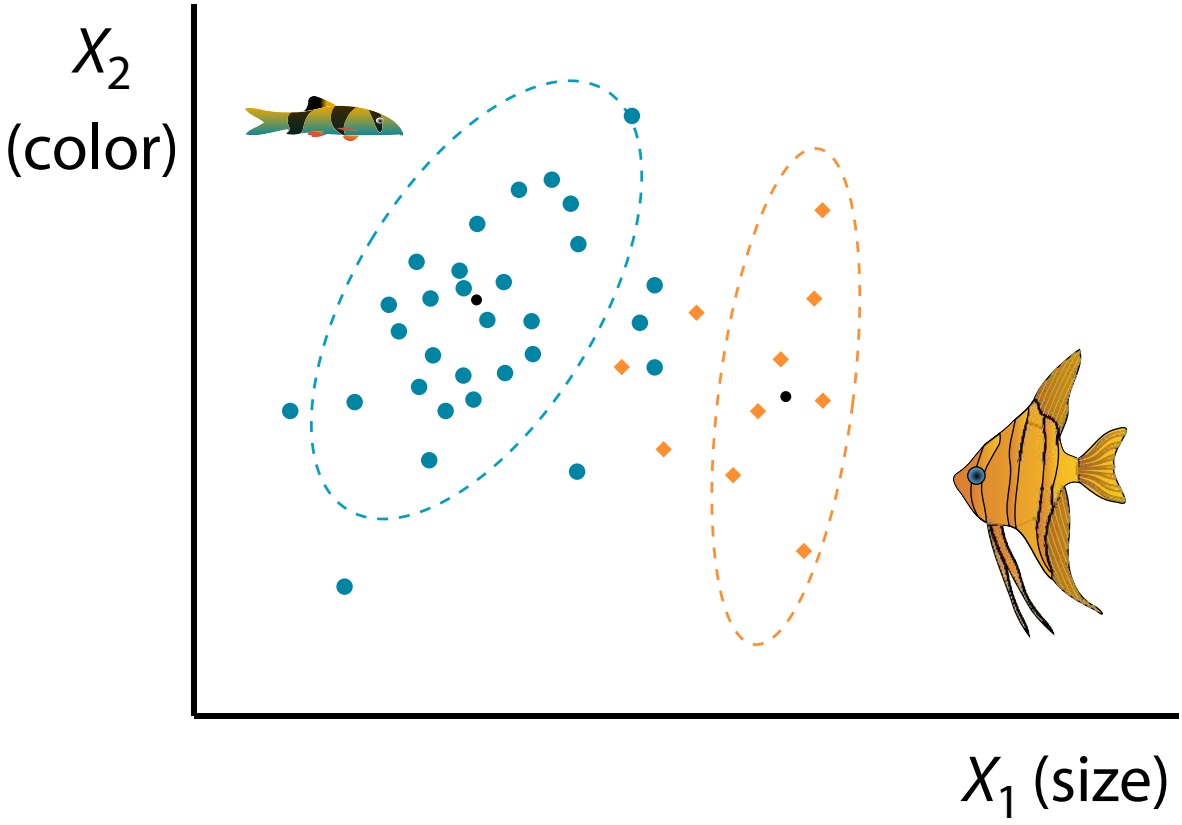

Figure 2.2: Distribution of little fish (circles) and big fish (diamonds)

maximum a posterior probability, which can be calculated by using Bayes' rule. In formula we write:

$$fish_{MAP} \equiv argmax\ P(fish \mid x) \tag{2.8}$$

$$= argmax\ \frac{P(x \mid fish)P(fish)}{P(x)} \tag{2.9}$$

$$\propto argmax\ P(x \mid fish)P(fish) \tag{2.10}$$

In the final step we drop the term $P(x)$ because it is a constant independent of $fish$. The $\propto$ means 'proportional to', as we are now no longer calculating the actual value of $P(fish \mid x)$ [Temperley2005]. We are left with just two terms, the term $P(fish)$ is the prior probability, the chance of the fish type before having observed any data point. We know we have a total of 40 fishes of which there are 30 little and 10 big fishes, therefore $P(fish = little) = \frac{30}{40} = \frac{3}{4}$ and $P(fish = big) = \frac{10}{40} = \frac{1}{4}$. Now all that is left to calculate is the likelihood term $P(x \mid fish)$. It can be calculated using the Gaussian density function, using the mean and covariance of the fish type we want to calculate the likelihood for.

Having calculated the a posterior probability for both fish types, we can determine which of the two types yields the highest and thus the maximum a posterior probability for the given data point x. This approach would work just as well if there were more than two types of fish, or of course in any other classification task.

2.2 Linear Dynamical System

Linear dynamical systems are used to model the behavior of a process over time, this process can be anything from the trajectory of an aircraft, to the number of fish each spring in a pond. We will discuss a *discrete* time linear dynamical system, which means that we model the change of the system from time t to time $t+1$, rather than modeling at a continuous rate of change. A linear dynamical system consists of at least two components, the *state vector* and the *transition matrix*. The state vector x_t completely describes the system at time t as shown below.

$$x_t = \begin{pmatrix} x_t^1 \\ x_t^2 \\ \vdots \\ x_t^n \end{pmatrix} \tag{2.11}$$

Each element $x_t^1, x_t^2, ..., x_t^n$ of the state vector holds some information regarding the current state of the system. The transition matrix A is an $n \times n$ matrix which contains the information to transform the state of the system at time t to the state of the system at time $t+1$. This is shown in formula 2.12 [Bretscher2001].

$$x_{t+1} = Ax_t \tag{2.12}$$

So the elements of the transition matrix tell us how we need to combine the various elements of the current state vector to get to the next state. The transition matrix is often chosen constant, but can also change with each time step.

2.2.1 Example

To get a better understanding of the working of linear dynamical systems we continue our example about the fishes. Being able to classify each fish we catch, we would like to study the population of the two fishes over time. To do this we will create a linear dynamical system which will tell us how the population changes each year because of new born fishes, little fishes being eaten by big fishes and big fishes dying from starvation. First we will need a state vector that holds all the information we want to monitor, as we are studying the population of the fishes our state vector simply consists of the number of big fishes and the number of little fishes, as shown below. For easy reference we shall refer to the big fishes as β and the little fishes as α.

$$x_t = \begin{pmatrix} \beta_t \\ \alpha_t \end{pmatrix} \tag{2.13}$$

Now we need to define a transition matrix, which will tell us how the population of each type of fish will grow each year. The big fishes grow each year according to the equation $\beta_{t+1} = 0.9\beta_t + 0.3\alpha_t$, while the little fishes grow according to the equation $\alpha_{t+1} = -0.2\beta_t + 1.6\alpha_t$. We can now enter these numbers in a transition matrix to arrive at the final form for our linear dynamical system, as shown below.

$$\begin{pmatrix} \beta_{t+1} \\ \alpha_{t+1} \end{pmatrix} = \begin{pmatrix} 0.9 & 0.3 \\ -0.2 & 1.6 \end{pmatrix} \begin{pmatrix} \beta_t \\ \alpha_t \end{pmatrix} \tag{2.14}$$

Using this system we can project ahead, to predict the population of fishes in the years to come. The results for this model based on the starting population of 10 big fishes and 30 little fishes are shown in table 2.1.

Table 2.1: Values of the predicted population of big fish (β) and little fish (α).

t	0	1	2	3	4
β_t	10	18	30	48	75
α_t	30	46	70	106	160

2.3 Kalman Filter

In the previous section we introduced linear dynamical systems to model the change of a process over time. Using this model we were able to predict the future outcome of the system. However, as no mathematical model is perfect, the predictions produced by this model are likely to deviate somewhat from the actual value. To get more accurate results we will include observations or measurements of the system we are modelling. Unfortunately measurements are not perfect either. In other words, both the linear dynamical system model and measurements are subject to noise. In this section we will discuss the Kalman filter, which is an optimal recursive data processing algorithm that will combine both the noisy model and noisy observations to estimate the actual state [Maybeck1979]. This estimation is done in a cyclic form, the Kalman filter first uses the model to predict the future state, this prediction is then corrected by incorporating the observation, finally the output from the correction is again used as input for the next prediction. This process is shown graphically in figure 2.3.

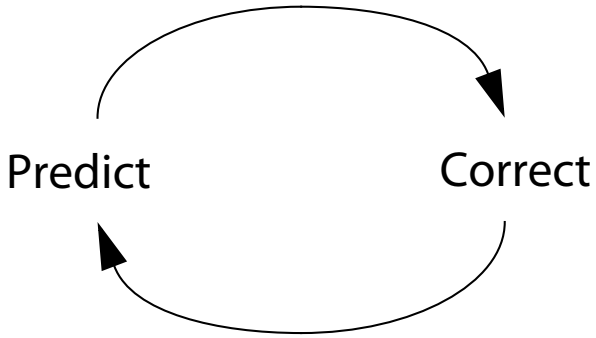

Figure 2.3: Kalman filter cycle, the filter predicts the upcoming state according to the linear dynamical system model and corrects the state by incorporating the observation

The Kalman filter models both the noise of the model and the observation as Gaussian white noise. Whiteness implies that the noise value is not correlated in time, while Gaussian means that it is a Gaussian random vector. We call the noise that affects the linear dynamical system model *system noise* and shall refer to it as w, its distribution has a mean of 0 and covariance matrix Q, in short $w \sim N(0, Q)$. System noise occurs during a state transition, when the system is going from one state to the other. The noise that affects the observation is called *measurement noise* and is referred to as v. It is a Gaussian random vector with a mean

of 0 and covariance R, in short $v \sim N(0, R)$. Measurement noise occurs during observations, when measuring the input variables. With these noise terms defined we can include them in the equation for the model and the observation. The model equation still describes the transition of state x_t to the next state, but now taking the noise into account. The observation equation describes the observation y_t as a linear combination of the observation matrix C and the state x_t. The observation matrix C describes which of the state variables can be observed.

$$x_{t+1} = Ax_t + w_t \tag{2.15}$$

$$y_t = Cx_t + v_t \tag{2.16}$$

The relationship between the states and the observations can be seen clearly in a graphical model known as a Dynamic Bayesian Network. This graphical model, shown in figure 2.4, represents the inner structure of the Kalman filter. The nodes in this graph represent Gaussian random variables and the arcs represent conditional dependence relations [MurphyWeb1998].

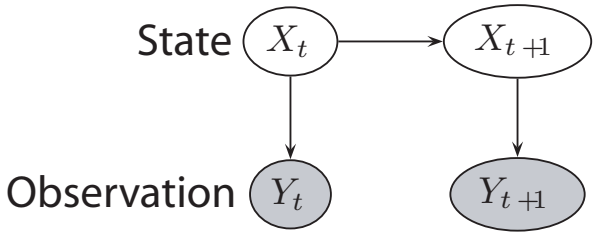

Figure 2.4: Dynamic Bayesian Network representation of Kalman filter model. The ellipses denote continuous nodes, where clear means hidden and shaded means observed.

To fully understand how and more importantly why the Kalman filter works, a derivation of its equations is necessary. This derivation can be found in appendix A, in this section we shall restrict ourselves to a simple description in words of how the filter works.

As mentioned earlier in this section, the filter works in a cyclic form where a prediction step is followed by a correction step. The prediction step is simply the projection to the future state by the linear dynamical system model as we saw in the previous section. The difference lies in that we are now representing the state as a Gaussian random vector, therefore next to the calculation of the next state the covariance of the next state is also calculated.

The true strength of the filter, however, lies in the correction step. Here the filter first calculates the Kalman gain matrix K, which can be seen as a sort of 'trust' matrix, expressing to what degree the observation and the prediction from the model should be trusted. This Kalman gain matrix is calculated by comparing the covariance of the observation with the covariance of the model prediction, while minimizing the covariance of the resulting filtered state [Welch2001].

The Kalman gain is used to correct the prediction of the model using the observation. All this can be seen in formula, having the same cyclic form as seen earlier, in figure 2.5. This figure also shows that the Kalman filter requires an initial state estimate to get it started. This initial estimate is entered at the calculation of the Kalman gain, meaning the filter starts its calculations at the correction step, correcting the initial estimate by incorporating the observation.

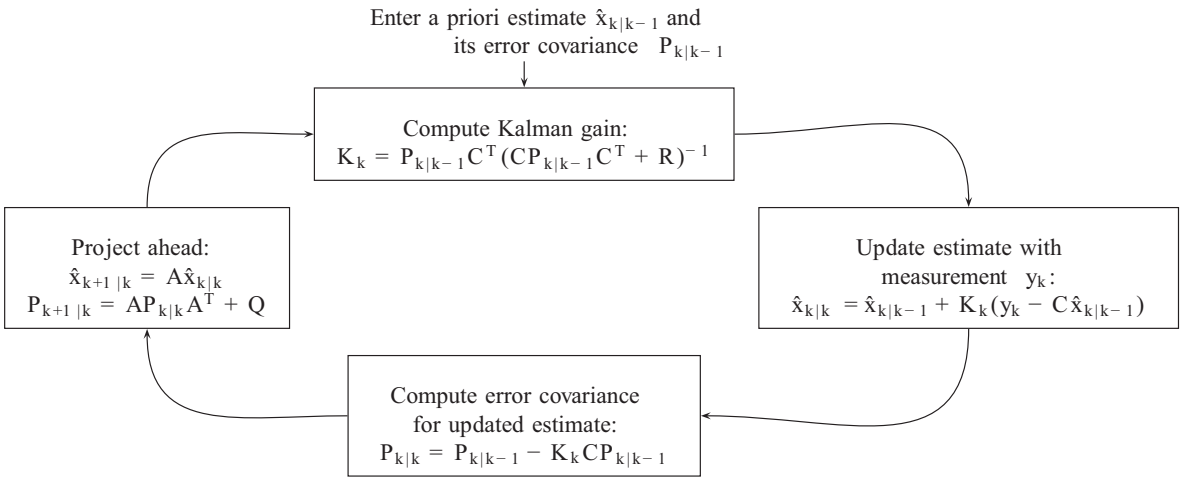

Figure 2.5: Kalman filter loop. (Taken from [Brown1992])

2.3.1 Example

We continue our example about the fish in the pond. In the previous section we saw how a linear dynamical system could predict the population of the fish in the years to come. Now we would like to use a Kalman filter to get more accurate results. Because the linear dynamical system model can impossibly take every effect on the population into account, the actual number of fish is likely to differ from what the model predicts. Therefore, we will each year take measurements from the pond, counting the population of fish on sight. Unfortunately the little fish are too small to count using this method, so we will only count the big fish and have the Kalman filter figure out the rest. Also, because fish are quite active creatures the measurement is not likely to be extremely accurate. We can enter all this information into the Kalman filter to get an optimal estimate on the population each year. The values of the variables in this example are shown in table 2.2, the observation y_t represents the counted number of big fish in year t, while the filtered estimates of the big fish and little fish are represented by β_t and α_t respectively, these results are rounded to the nearest integer value.

Table 2.2: Values of the estimated population of big fish (β) and little fish (α).

t	0	1	2	3	4
y_t	10	16	27	49	71
β_t	10	17	28	48	72
α_t	30	45	66	108	155

2.4 Summary

In this chapter we explored some basic mathematical principles and examined simple examples to illustrate their working. We have discussed probability theory and have seen how we can use Bayes' rule for inference. We then looked at how we can use a linear dynamical system to predict future states of a process being modelled. Finally we discussed how we can use the

Kalman filter by combining a linear dynamical system with observations to get more accurate results of the process being modelled. With these mathematical principles explained, we can now introduce the variables needed for setting up a linear dynamical system to model the tempo in music. This will be the focus of our next chapter, we will introduce the variables of interest piece by piece ending up with a complete model that can be used for tempo tracking.

Chapter 3

Mathematical model

In this chapter we will introduce a model for tempo tracking using the mathematical principles discussed in the previous chapter. In order to create such a model we first need to quantize some features of music, this will allow us to perform calculations on them. We will introduce various variables and show their relationship among each other, leading to a linear dynamical system model. The basis of our quantization will be a piece of music score, this music score is no different from sheet music that musicians use to know what notes they have to play and how long they have to play them.

Our model will use the onset time of notes as input, therefore we are not interested in the pitch of a note (the height of a note), making the examples in this chapter sound rather dull when played on the piano. However, the sole purpose of these examples is to illustrate the topic being discussed.

In sections 3.1 to 3.4 we will introduce various variables needed to create a mathematical model for tempo tracking. Section 3.5 then explains how these variables are combined to form a linear dynamical system. Finally, we introduce the notion of hidden variables in section 3.6 and explain what we can observe, completing our mathematical model for tempo tracking.

3.1 Score Positions

The piece of music score in figure 3.1 is the same score we saw in the introduction section. However, it now includes the index of each note at the top, and the score positions of each note at the bottom.

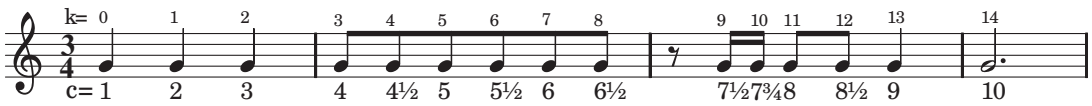

Figure 3.1: A piece of music score with index and score positions

The score position of a note is basically the same as the count at which the note occurs, but instead of starting over at 1 every time a measure ends, the counting continues. As we

are going to use terms such as the score position in mathematical equations, it is convenient to introduce some symbols. We shall refer to the index of a note as k, where the first note has the value of $k = 0$. The score position of a note is referred to as c [Cemgil2004]. For example, in figure 3.1 we see that $c_0 = 1$ and $c_{10} = 7\frac{3}{4}$. An important thing to notice in the figure is that the rest in between the notes at $k = 8$ and $k = 9$ neither has an index nor a score position, because there is no note being played at the start of a rest, there will not be an input signal at this point either, therefore it is ignored by the model.

3.2 Score Difference

The next term we will introduce is the score difference, referred to as γ [Cemgil2004]. In figure 3.2 again the index of each note is shown at the top, but now the score difference is shown at the bottom, both index and score difference are shifted a little bit to the left to clearly show the difference between which two notes we are using.

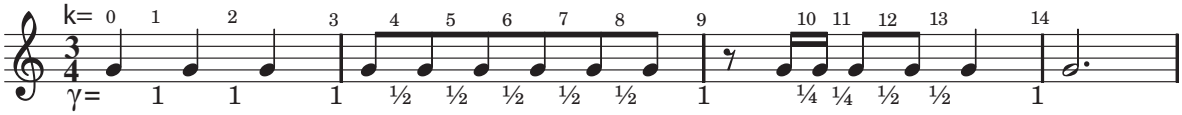

Figure 3.2: A piece of music score with shifted index and score differences

As the term suggests, the score difference is the difference between two consecutive notes. This means the score difference γ_k refers to the difference between the kth note and the $k-1$st note. For this reason, the first note in a piece of score ($k = 0$) does not have a score difference value. This approach is used because when we observe a note, we can only determine its difference once the next note has been observed.

As the index and score difference in figure 3.2 are shifted to the left it is not entirely clear which note matches with which index, we therefore show the same values shifted back to the right in figure 3.3. This shows how the score differences are stored together with a certain note, just keep in mind that the score difference is between the kth note and the $k-1$st note.

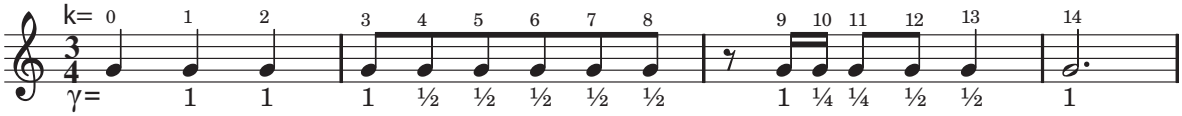

Figure 3.3: A piece of music score with index and score differences

When looking closely at figure 3.3, it looks as if the score difference γ_k is the same as the note duration for note $k-1$, where note duration refers to how long a note lasts. For example, the note at $k = 2$ is a quarter note and thus its duration is one count. The score difference value at note $k = 3$ is exactly this one count. As you can see this applies to almost all the notes in the figure, except for the notes at $k = 8$ and $k = 9$. This is caused because of the rest, because the rest is ignored by the model the difference between the two notes is bigger than the note duration. This applies even more so when dealing with polyphonic music. When comparing figure 3.1 with 3.3 we see that the score difference γ can be calculated from the

score position c, namely by $\gamma_k = c_k - c_{k-1}$. We can illustrate this dependence relationship in a graph as shown in figure 3.4. We will later expand this graph so that it will show our entire model.

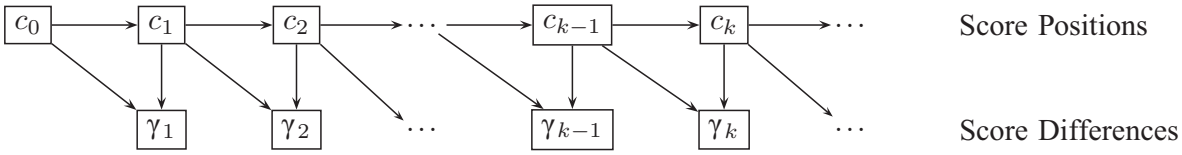

Figure 3.4: Graphical model illustrating the dependence relationship between score position c and score difference γ. (taken from [Cemgil2004])

3.3 Onset time

The terms of score position and score difference are discrete versions of what we are actually dealing with, namely time. Everything happens at a certain moment in time, we call such a moment a timestamp, expressed in seconds. The moment a note is played also has its own timestamp, we refer to this particular timestamp as the onset time. Each note that is played can be expressed using an onset time and an offset time, which is the time the note started and the time it stopped, respectively. In our model for tempo tracking we will only use the onset time of a note and shall refer to it as τ [Cemgil2004]. Figure 3.5 shows the piece of score we have been using as an example thus far with computer generated onset times, meaning we used a constant tempo and the onset times were generated with millisecond accuracy.

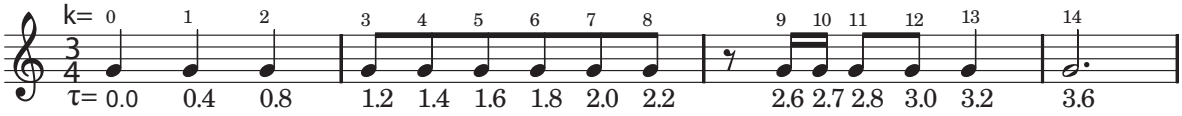

Figure 3.5: A piece of music score with index and computer generated onset times

3.4 Period

Next we will want to introduce a term for tempo, as that is what we are after, tempo is often denoted in beats per minute or bpm for short. Say we have a value of 60 bpm, then this means there will be sixty beats within a minute, this is the same as saying that a single beat occurs every second, or that the time between two consecutive beats lasts one second. The time between two consecutive beats can also be referred to as the period, and that is the term we will use, we will refer to the period as Δ [Cemgil2004]. It should be noted that the period is *not* the equivalent of score difference γ on a continuous scale. This is because the period is always the time between two consecutive *beats*, not the time between two consecutive notes. We could, however, calculate the time between two consecutive notes by combining both the period Δ and the score difference γ. If we then add the result of this calculation to the previous onset time, we will have calculated the new onset time. In formula this gives:

$$\tau_k = \tau_{k-1} + \gamma_k \Delta_{k-1} \tag{3.1}$$

It can be seen clearly that this formula works when looking at figure 3.6. The onset times in this figure were generated using a constant period of 0.4 seconds, which is equivalent to 150 bpm. The arrows above the notes indicate the position of the beat, so we can clearly see the time between two arrows always adds up to 0.4, while the time between two notes depends on the value of the score difference.

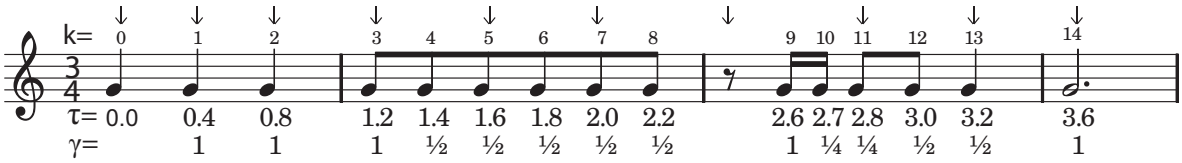

Figure 3.6: A piece of music score with index, computer generated onset times and score difference

With these new terms defined we can expand our previous graphical model to include the dependence relationships from our newly created formula 3.1, this gives a new graph shown in figure 3.7.

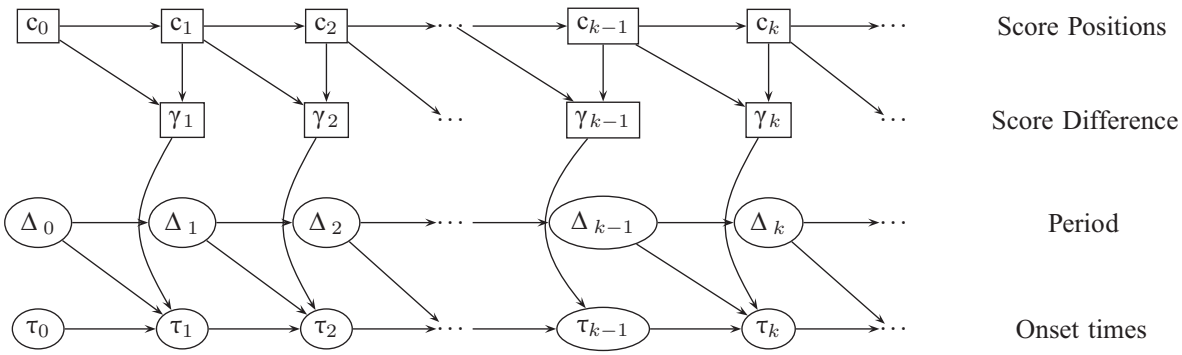

Figure 3.7: Graphical model illustrating the dependence relationship between score position c, score difference γ, period Δ and onset time τ. The square nodes denote discrete random variables and the oval nodes denote continuous random variables. (taken from [Cemgil2004])

3.5 Linear Dynamical System

We have now defined enough variables to be able to create a linear dynamical system. As discussed in section 2.2, where we introduced linear dynamical systems as a mathematical concept, we will first have to determine how to define our state vector x. A state vector completely describes the system at a certain moment in time, as we are creating a generative model for onset times, the state vector should include the onset time and the period. The state vector will therefore be defined as follows:

$$x_k = \begin{pmatrix} \tau_k \\ \Delta_k \end{pmatrix} \tag{3.2}$$

To complete our linear dynamical system, we also have to define the values of the transition matrix A. With the earlier defined formula $\tau_k = \tau_{k-1} + \gamma_k \Delta_{k-1}$ in mind, together with our defined state vector, we can simply fill in the blanks. Assuming a constant tempo, our transition matrix will be $A = \begin{pmatrix} 1 & \gamma \\ 0 & 1 \end{pmatrix}$ and thus we get the following model for our linear dynamical system [Cemgil2004].

$$\begin{pmatrix} \tau_k \\ \Delta_k \end{pmatrix} = \begin{pmatrix} 1 & \gamma_k \\ 0 & 1 \end{pmatrix} \begin{pmatrix} \tau_{k-1} \\ \Delta_{k-1} \end{pmatrix} \tag{3.3}$$

Using this model we can generate the onset times of notes produced by a certain score, of course we would need to know the score difference between the notes and the initial state, consisting of the onset time and period. However, this is actually not a very realistic model for music as music is full of intentional and unintentional deviations from the actual score. For example, a musician could decide to play a certain note just a little bit later than the score dictates to create a rhythmic effect. On the other hand it is simply impossible for a human being to play with such exact precision that every quarter note (or any other note) lasts exactly the same number of milliseconds. These deviations cause effects in the music, which can make it more alive and beautiful. However, as these deviations corrupt the exact data that we are after, to us they are no more than noise. We should take this noise into account, which is why we add system noise to our model. We learned about system noise in section 2.3 when we introduced the Kalman filter, it is Gaussian white noise with a distribution $w \sim N(0, Q)$. This gives us the final model for generating onset times [Cemgil2004].

$$\begin{pmatrix} \tau_k \\ \Delta_k \end{pmatrix} = \begin{pmatrix} 1 & \gamma_k \\ 0 & 1 \end{pmatrix} \begin{pmatrix} \tau_{k-1} \\ \Delta_{k-1} \end{pmatrix} + w_k \tag{3.4}$$

3.6 Hidden variables

When examining the model we have obtained thus far, it shows that we need both the onset time τ and the period Δ to be able to work with our model. However, as we will see in this section both these variables are hidden variables, which means they cannot be observed directly. We cannot observe the period directly because it is something which can only be measured within a certain timespan. We cannot observe the actual onset time either, but we can observe a noisy version of the onset time. This noise is called measurement noise, which we discussed earlier in section 2.3, it is Gaussian white noise with a distribution $v \sim N(0, R)$. We can describe the observation in a mathematical equation of the form $y_k = Cx_k$, where x_k is the state vector, C the observation matrix and y_k the observation at time k [Murphy1998]. The observation matrix holds the information on what variables in the state vector can actually be observed. As in our case only the onset time is observed, for us this formula looks as follows [Cemgil2004].

$$y_k = \begin{pmatrix} 1 & 0 \end{pmatrix} \begin{pmatrix} \tau_k \\ \Delta_k \end{pmatrix} + v_k \tag{3.5}$$

This gives us our final version of our dependency graph, shown in figure 3.8 and completes our mathematical model.

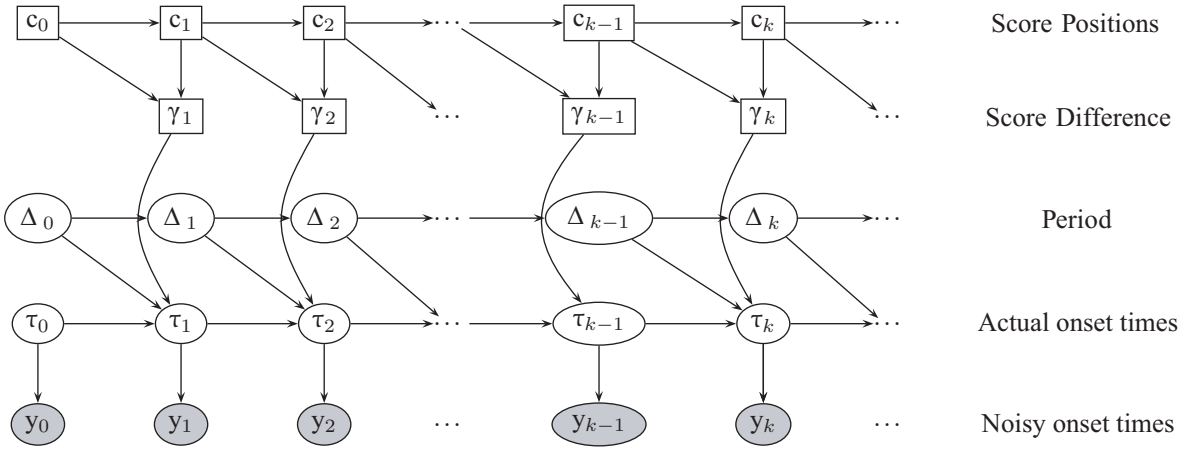

Figure 3.8: Graphical model illustrating the dependence relationship between score position c, score difference γ, period Δ, actual onset time τ and noisy onset time y. The square nodes denote discrete random variables and the oval nodes denote continuous random variables, where clear means hidden and shaded means observed. (taken from [Cemgil2004])

3.7 Summary

In this chapter we learned how we can use the score position c and the score difference γ to describe the score of a musical piece. We then introduced the onset time τ and period Δ and learned how we can combine the two to form a linear dynamical system for generating onset times for a matching score. Finally, we saw that the onset time and period are actually hidden variables, and how we can only observe a noisy version of the onset time which we refer to as y.

Because we do want to know the actual onset time τ and the period Δ, we will use the Kalman filter to get as good an estimate of those two values as possible. We have discussed the principles of the Kalman filter in section 2.3. In the next chapter we will further expand these principles and apply them to our mathematical model created. This will result in a working algorithm for doing tempo tracking.

Chapter 4

Methods

In the previous chapter we created a linear dynamical system model for generating onset times for a matching music score. We learned that the state vector of this model is hidden, creating the need for us to estimate the value of the state. In this chapter we will describe methods based on the Kalman filter to perform this estimation. We have already discussed the basics of the Kalman filter that was introduced by R.E. Kalman in 1960 [Kalman1960]. However, his method requires some expansions in order to work for tempo tracking.

In this chapter we will discuss these expansions and will illustrate them with examples of tempo tracking scenarios. In section 4.1 we will introduce the switching Kalman filter, a method that allows us to combine a number of linear dynamical systems while still using the Kalman filter for estimation. We will illustrate the use of the switching Kalman filter by giving an example of tempo tracking on a musical performance of which the score is known in advance. In section 4.2 we will introduce the particle filter, a method in which several hypotheses, referred to as particles, are tested to determine which fits a certain scenario best. With this method we will be able to perform tempo tracking on a musical performance with unknown score, as will be shown in an example.

4.1 Switching Kalman filter

A restriction in applying the Kalman filter is that it operates only on linear models. Unfortunately, most systems are not linear and thus their state cannot be estimated using the Kalman filter. One solution to this restriction is to use a piecewise linear model, the model as a whole is not linear but the individual pieces that make up the model are. We will have to create a list of the different linear models that make up the piecewise linear model and switch between the different models as time progresses. To make this work in combination with the Kalman filter, we will need a discrete switch variable S_t that specifies which model to use at time t [Murphy1998]. Using this switch variable we can replace the model parameters in the Kalman filter with the ones needed for that particular time step. This expanded form of the Kalman filter is shown graphically in figure 4.1.

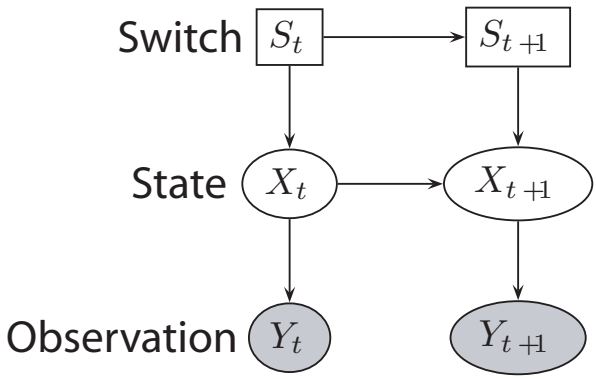

Figure 4.1: Dynamic Bayesian Network representation of switching Kalman filter model. The circles denote continuous nodes, where clear means hidden and shaded means observed.

4.1.1 Applied to Tempo Tracking

The mathematical model that we created in the previous chapter actually lends itself perfectly for a switching Kalman filter. Our linear dynamical system model uses a transition matrix that contains a variable, namely the score difference γ_k. This means that depending on the value of this score difference γ_k we will be using a different linear dynamical system model every time this value changes. As score difference is something that can change with each time step it is exactly the switching variable that we need. For clarities sake we show the transition matrix A one more time below.

$$A_k = \begin{pmatrix} 1 & \gamma_k \\ 0 & 1 \end{pmatrix} \tag{4.1}$$

So in the case of our tempo tracking model the only thing that needs to be changed when switching to a different linear model, is the transition matrix A. We can see this even more clearly by mapping the graphical model of the switching Kalman filter to the graphical model we created in the previous chapter, the result is shown in figure 4.2.

4.1.2 Example

Having discussed the switching Kalman filter and how it applies to tempo tracking, we can have our first attempt at performing tempo tracking on actual human generated musical data. In this section we will discuss how we apply the switching Kalman filter by going over each step in detail and discuss the output that the filter generates.

Before Starting with this example we have to introduce a restriction. In using the switching Kalman filter we need a switch variable, we already decided that the score difference γ would make a good switch variable. However, this means that we will need to know the value of the score difference in advance, otherwise we will not know which linear model to use. This means that we will need to know the exact score of the piece we are doing the tempo tracking on in advance, so we will have to manually input the score difference values before we can start using the switching Kalman filter. The score for this example together with the noisy onset time observations y and score difference γ are shown in figure 4.3.

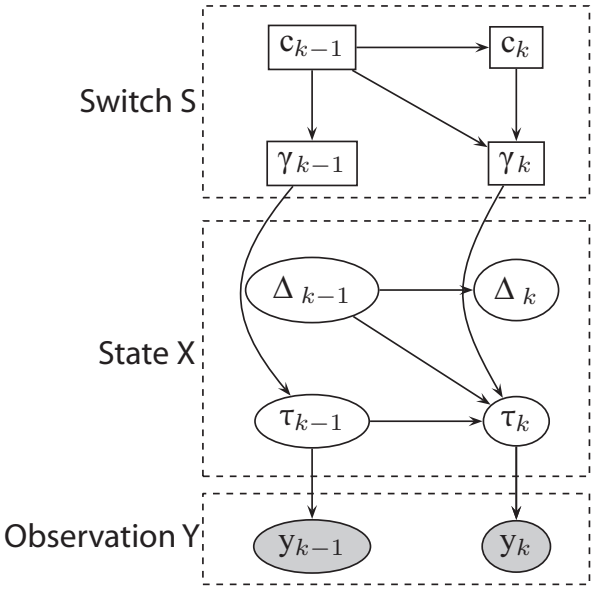

Figure 4.2: Mathematical model for tempo tracking with highlighted elements for a switching Kalman filter model. The graph shows score position c, score difference γ, period Δ, actual onset time τ and noisy onset time y. The square nodes denote discrete random variables and the oval nodes denote continuous random variables, where clear means hidden and shaded means observed.

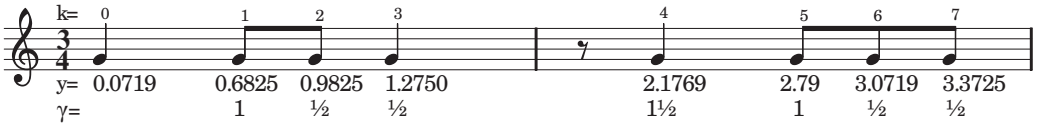

Figure 4.3: Known score with noisy onset time observations y and score difference γ

To be able to use the Kalman filter we need to initialize a number of parameters. We have already defined the transition matrix A and the observation matrix C when we discussed the mathematical model in chapter 3. What we still need to define is the error covariance matrix Q for the system noise and the error covariance matrix R for the measurement noise, we will use constant values for these matrices instead of having them change at every timestep. The exact values for Q and R will be chosen somewhat ad hoc, they can be determined in a sensible way using the Expectation-Maximization(EM) algorithm, but for this simple example it will suffice to just use logical reasoning. The process noise covariance Q describes how certain we are about the results of our model, setting this value to 0 would indicate absolute certainty, while giving it a large number would indicate the model is close to just making random guesses. Because we have a pretty strict model, we will choose a rather low value showing we are quite certain but allow some fluctuation. This fluctuation is needed because the tempo is modelled as a constant value in our model, which is obviously not the case. Because our state vector consists of two variables (the onset time and period) our matrix Q will be a 2×2

matrix.

$$Q = \begin{pmatrix} 0.001 & 0 \\ 0 & 0.001 \end{pmatrix} \tag{4.2}$$

The measurement noise covariance R describes how certain we are about the measurement we have obtained. Our measurements were obtained using a MIDI keyboard, so the measurement of the keypresses will be near perfect. However, because there was a human being playing the keyboard we can assume there will be some deviations from the score, we will therefore choose the value of R as 0.01.

Next to the covariance matrices we need to determine our initial a priori state vector and covariance matrix, which is required for the Kalman filter as a starting point. As we are unable to determine these for sure, we will assume music performances usually start immediately, so at a timestamp of 0.0 seconds. For the period we will choose a value of 1.0, which is equivalent to 60 bpm. Because we really do not know for sure what the initial state is, we will choose a high value for the covariance matrix. We therefore get the following results for our initial state.

$$\hat{x}_0^- = \begin{pmatrix} \hat{\tau}_0^- \\ \hat{\Delta}_0^- \end{pmatrix} = \begin{pmatrix} 0 \\ 1 \end{pmatrix} \qquad P_0^- = \begin{pmatrix} 1.0 & 0 \\ 0 & 1.0 \end{pmatrix}$$

When we are referring to the a priori state, which is the result of the prediction step, we write $\hat{x}_{k|k-1}$, meaning we are referring to the estimate at point k given the previous state. The a posteriori state, which is the result of the correction step, is referred to as $\hat{x}_{k|k}$, because after having performed the correction step the observation at time k has been incorporated in the result. For the initial state, shown above, we used a super minus sign to indicate it being the a priori state, as it starts at time $k = 0$ and there is no previous state it is conditioned on.

We now have all the parameters initialized to start processing the noisy onset observations. If a review of the inner working of the Kalman filter is required, please refer to figure 2.5, which shows the cyclic form of the filter together with the formulas used in the algorithm. We enter the filter at the correction step, which will correct our initial state by incorporating the observation. When performing the necessary calculations, it will show that the result of the correction step is as shown below.

$$\hat{x}_{0|0} = \begin{pmatrix} \hat{\tau}_{0|0} \\ \hat{\Delta}_{0|0} \end{pmatrix} = \begin{pmatrix} 0.0712 \\ 1 \end{pmatrix} \qquad P_{0|0} = \begin{pmatrix} 0.0099 & 0 \\ 0 & 1.0 \end{pmatrix}$$

We see from these results that the estimated onset time is now almost similar to the observation y_0 which was 0.0719. This makes perfect sense as we gave our initial a priori state a very high covariance, indicating we were not certain of its values at all. We also see that the a posteriori covariance for the onset time has become a lot smaller, indicating a much higher certainty regarding this value. On the other hand we see both the value of the period and its covariance unchanged, this too makes perfect sense. It makes sense intuitively, because the period is something which is measured within a certain timespan and so far we have only observed one onset time. It also makes sense when examining the numbers, we initialized our initial a priori covariance matrix P_0^- as a diagonal matrix, thus saying that the onset time and period are uncorrelated, furthermore our observation matrix says we can only observe the onset time and not the period, the Kalman filter therefore had no way of telling anything about the period and had to rely fully on the initial value.

Having obtained the filtered estimate for $k = 0$, we can continue to $k = 1$. The score difference for this point is $\gamma_1 = 1$, so we can switch to a linear dynamical system model in which the transition matrix A_1 is initialized with a value of $\gamma_1 = 1$. Now we can start the prediction step and calculate the projection to the next state according to our model, this gives us the following a priori state with matching covariance.

$$\hat{x}_{1|0} = \begin{pmatrix} \hat{\tau}_{1|0} \\ \hat{\Delta}_{1|0} \end{pmatrix} = \begin{pmatrix} 1.0712 \\ 1 \end{pmatrix} \qquad P_{1|0} = \begin{pmatrix} 1.0109 & 1.0 \\ 1.0 & 1.001 \end{pmatrix}$$

As we entered the prediction step with a period of 1 it was simply added to the onset time, just like our model says. Also, the model updated the covariance matrix now showing that the onset time and period actually are correlated, continuing with these results to the correction step, we get the filtered estimates for $k = 1$.

$$\hat{x}_{1|1} = \begin{pmatrix} \hat{\tau}_{1|1} \\ \hat{\Delta}_{1|1} \end{pmatrix} = \begin{pmatrix} 0.6863 \\ 0.6193 \end{pmatrix} \qquad P_{1|1} = \begin{pmatrix} 0.0099 & 0.0098 \\ 0.0098 & 0.0215 \end{pmatrix}$$

Again we see that the filtered onset time lies closely to the observed noisy onset y_1 of 0.6825 and not close to the prediction that the model gave. However, this time the period was updated as well because the filter was now able to tell how the onset time and period are correlated. The covariance matrix also shows a lower covariance for the period indicating a higher certainty.

We can now proceed to $k = 2$, the score difference for this point is $\gamma_2 = \frac{1}{2}$, we therefore switch to a linear dynamical system model in which the transition matrix A_2 is initialized with a value $\gamma_2 = \frac{1}{2}$. Because we switched to a different linear dynamical system model we will go over the results from the filter in detail one more time, so we can clearly see the effect of the switch. The results from the prediction step are as follows.

$$\hat{x}_{2|1} = \begin{pmatrix} \hat{\tau}_{2|1} \\ \hat{\Delta}_{2|1} \end{pmatrix} = \begin{pmatrix} 0.9959 \\ 0.6193 \end{pmatrix} \qquad P_{2|1} = \begin{pmatrix} 0.0261 & 0.0205 \\ 0.0205 & 0.0225 \end{pmatrix}$$

We see now that the onset time was updated with only half a period, as the score difference for this timestep was $\gamma_2 = \frac{1}{2}$. Also because the period was updated in the previous step, the model prediction is a lot more accurate, as we will see after having performed the correction step, whose results are listed below.

$$\hat{x}_{2|2} = \begin{pmatrix} \hat{\tau}_{2|2} \\ \hat{\Delta}_{2|2} \end{pmatrix} = \begin{pmatrix} 0.9862 \\ 0.6116 \end{pmatrix} \qquad P_{2|2} = \begin{pmatrix} 0.0072 & 0.0057 \\ 0.0057 & 0.0108 \end{pmatrix}$$

We see that the filtered onset time lies somewhat in between the predicted onset time and the observed onset time y_2 of 0.9825, meaning the filter is starting to trust the prediction about as much as the observation. This is not that strange as the two values start to differ less and less, in other words the results from the model start to agree more and more with the observations meaning we must be on the right track, or tempo.

For the remaining steps we continue to initialize the transition matrix to the matching γ_k for that step and process the measurement. The results can be found in table 4.1, only the filtered state estimate is given and no longer the a priori estimate, as it is the filtered

Table 4.1: Values of filtered state estimates for example with a known score

k	0	1	2	3	4	5	6	7
y_k	0.0719	0.6825	0.9825	1.2750	2.1769	2.79	3.0719	3.3725
$\hat{\tau}_{k\|k}$	0.0712	0.6863	0.9862	1.2814	2.1794	2.7866	3.0798	3.3765
$\hat{\Delta}_{k\|k}$	1.0000	0.6193	0.6116	0.6045	0.6009	0.6031	0.6002	0.5988

values that we are after. Furthermore a plot of the onset times and period together with their covariance can be found in figure 4.4. The X axis denote the onset time, while the period is plotted on the Y axis, the ellipses represent the covariance matrix at each timestep.

From the plot and the values in the table we can clearly see that the period quickly converges to a value of around 0.6, the plot also shows how the initial period of 1.0 is adjusted after one iteration. Furthermore, we see how the ellipses in the plot become smaller and smaller, the ellipses represent the covariance matrices. The fact that the values of the covariance matrix are decreasing means that the uncertainty is decreasing as well. Although the performance in this example was done by a human being, the notes were played at a rather constant tempo. The filter would, however, have no trouble tracking a more irregular tempo.

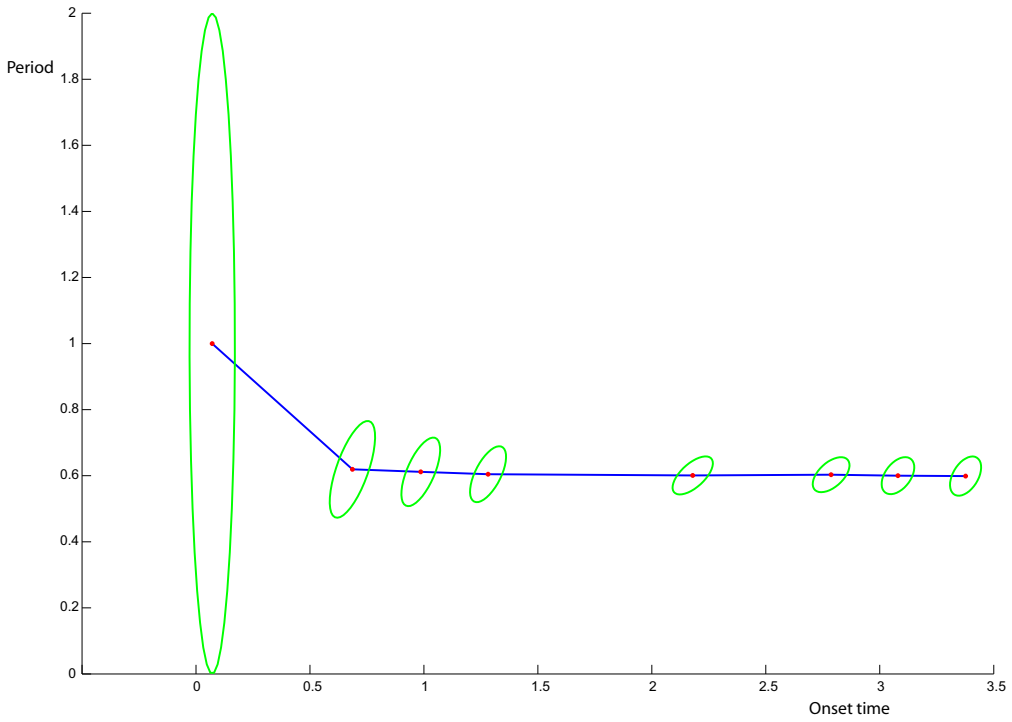

Figure 4.4: Plot of filtered onset times (x axis) and period (y axis) together with covariance for example with known score.

4.2 Particle Filter

In the previous section we discussed the switching Kalman filter in which a switching variable indicated which linear dynamical system should be used. We saw an example of tempo tracking in which a switching Kalman filter was used to track the tempo of a performance of which the score was known in advance. Because the score is not always known in advance, we would also like to be able to do tempo tracking on a performance with unknown score. In other words, we need a method that allows us to use the switching Kalman filter while the switching variable is hidden. In this section we will discuss such a method known as the particle filter.

A particle filter can be used to determine the value of the switch variable in a switching Kalman filter, without the value of the switch variable we have no way of telling which linear model to use and can therefore not perform the necessary calculations for the switching Kalman filter. The approach that the particle filter takes in solving this problem is to simply try all possible values as hypotheses for the switch variable, these hypothetical values are referred to as particles. For each of these particles the calculations of the switching Kalman filter will be performed, giving us a collection of potential states for the next timestep.

To differentiate between these states the likelihood of each of them is calculated, the likelihood is a probabilistic term indicating how likely the observation is given that it is interpreted using a particular switching variable. Next to the likelihood we can also include a prior probability, the prior probability indicates a preference for a certain switching variable independent of the observation made. The use of a prior probability can be required as it is possible that multiple solutions receive the same likelihood, the prior probability will allow use to differentiate between those solutions. We can combine the likelihood and the prior probability by using Bayes theorem giving us a result that is proportional to the posterior probability. We can then determine which state has the highest proportional posterior probability and use it to continue to the next timestep.

$$p(class \mid data) \propto p(data \mid class) \times P(class) \tag{4.3}$$

$$posterior \propto likelihood \times prior \tag{4.4}$$

Instead of just using the particle that resulted in the state with the highest posterior probability, we would like to keep track of the other particles as well. A particle that seemed plausible at first could turn out to result in a lot less likely particles after a few iterations. We can use other particles as parents for a new iteration too, this results in a tree like structure where each route through the tree is a sequence of switching variable values. But using all particles means the tree grows exponentially with each iteration resulting in an explosion of particles within a few of iterations, this will make the number of calculations too large and thus the algorithm too slow, especially for realtime applications. Therefore, as a pruning condition we will only use the n-best particles as parents for the new iteration, this will keep the number of calculations constant while still maintaining an interesting set of particles. Also, instead of using all possible values as hypotheses for the switching variable, we could decide to select only a few values that seem to make sense, this could even be required as the list of possible values can be very large or even infinite.

Using the particle filter we can get the estimation of the state as well as the value of the switching variable, the method can be applied in offline and online scenarios and can thus be used for realtime processing as well.

4.2.1 Applied to Tempo Tracking

With the principles of the particle filter explained we can now discuss how to use it in tempo tracking. We introduced the particle filter because in our previous approach we were restricted to a scenario in which the score of a musical performance was known in advance. Using the particle filter we will also be able to process performances with unknown score, we therefore need to create a set of possible switch variables and define a method for calculating the likelihood.

The model that we used when working with the switching Kalman filter in the previous example still applies, this means that our switching variable is still the score difference. As the score difference is not known, the particle filter will try every possible value to determine which one works best. This means we need to know what possible values a score difference can take and we need to come up with a calculation for the likelihood of such a value. The list of possible score differences is basically infinite, a musician can have a note last as long as he likes or as short as he likes, meaning the score difference can be almost any value from 0 to infinite. But because the score difference is a discrete random variable, it has to stick to a certain grid of possible values. To be able to use the particle filter we will have to make a selection from this infinite list of values, fortunately this is possible. Roughly put, when the difference in time between two notes is very small we can ignore the very high score differences as it very unlikely they will apply, similarly if the difference in time is very large we can ignore the very small score differences.

Put more exact, we use the score difference to be able to get a prediction for the next timestep from the linear model, however, we also have a measurement for the next timestep. The difference between the measurement and the prediction from the model is referred to as the residual e_k, it is calculated by $e_k = y_k - Cx_{k|k-1}$ and reflects the discrepancy between the predicted estimate $Cx_{k|k-1}$ and the measurement y_k. Ideally this residual is equal to zero meaning the model and observation are in complete agreement, we can calculate with which score difference this residual will be zero by performing some simple algebra, as shown in the following derivation.

$$
\begin{aligned}
e_k = y_k - Cx_{k|k-1} &= 0 \\
y_k - CAx_{k-1|k-1} &= 0 \\
y_k - \begin{pmatrix} 1 & 0 \end{pmatrix} \begin{pmatrix} 1 & \gamma_k \\ 0 & 1 \end{pmatrix} \begin{pmatrix} \tau_{k-1} \\ \Delta_{k-1} \end{pmatrix} &= 0 \\
y_k - \tau_{k-1} - \gamma_k \Delta_{k-1} &= 0 \\
\frac{y_k - \tau_{k-1}}{\Delta_{k-1}} &= \gamma_k
\end{aligned}
$$

According to this derivation the ideal score difference can be calculated by taking the difference between the observation and the previous onset time and dividing it by the period. The result from this calculation will be a real number, but because we are modelling the score difference as a discrete random variable we will need to round this real number to the nearest value on our grid of discrete values. To prevent working with really short notes we will choose a smallest note representation, meaning we will choose a minimum value for the resolution of our grid of discrete values. This idea is illustrated by a simple fictive example in figure 4.5, say we have calculated that the ideal score difference is 1.34, we have chosen a grid resolution

of 0.25 the score difference that will be used will be 1.25. Next to this value we can use a number of its neighbouring grid values as well to get a list of potential score differences.

Figure 4.5: Fictive example, calculated ideal score difference is 1.34 this is rounded to the nearest position on a grid with a resolution of 0.25

Having developed a method to create a subset from the set of all score differences, our next step is to find a method for calculating the likelihood of each of these values. The ideal scenario is that the linear model and the observation are in complete agreement, we have already seen that the residual is the difference between the prediction by the linear model and the observation, so the residual would make a good basis for our calculation of the likelihood. We will calculate the likelihood by sampling from a Gaussian distribution $N(0, covE)$. The mean of this distribution is 0 as the ideal residual is 0 as well, while the covariance $covE$ is calculated as $covE = CP_{k|k-1}C^T + R$. The likelihood $p(e_k \mid \mu, \Sigma)$ is then calculated using the formula to calculate the probability of a sample from a Gaussian distribution as discussed in section 2.1.6.

Next to the likelihood we could also define a prior probability, as mentioned before a prior probability indicates a preference for a certain switching variable independent of the observation made. As sometimes multiple solutions achieve the same likelihood, this prior can be used to express a preference. In tempo tracking we are trying to determine both the period and score difference of a musical performance, together these two values should match the observed onset times. However, there are various combinations of the period and score difference that give the same exact match, we can simply scale the two variables by a certain factor. This can be seen clearly in the example shown in figure 4.6, we have a list of onset times y and want to find the right combination of period Δ and score difference γ. Interpretation 1 models the notes as quarter notes with a period of 1.0 (which is equivalent to 60 bpm), this gives a perfect match to the given list of onset times, which in turn would result in the highest likelihood possible according to our just defined formula for calculating the likelihood. Looking at interpretation 2 we see the notes are modeled as half notes with a period of 0.5 (equivalent to 120 bpm), giving also a perfect match and thus the exact same likelihood. As the figure shows, the two implementations simply differ by a scaling factor two (score difference times two, period divided by two).

We could scale the solution in all sorts of ways as long as there is a score difference to match it, this could give us very strange solutions with very uncommon score representations as a result. To filter out these uncommon scores we can include a prior probability, we could for example say we favor quarter notes over any other note by giving the quarter note a higher prior probability than the other notes. The problem in music is that there is not really a ground truth that holds for all musical genres, it is therefore difficult to define a prior probability that will work well for every musical performance. A solution to this might be to define several prior probabilities for different musical genres and have the user select a genre in advance.

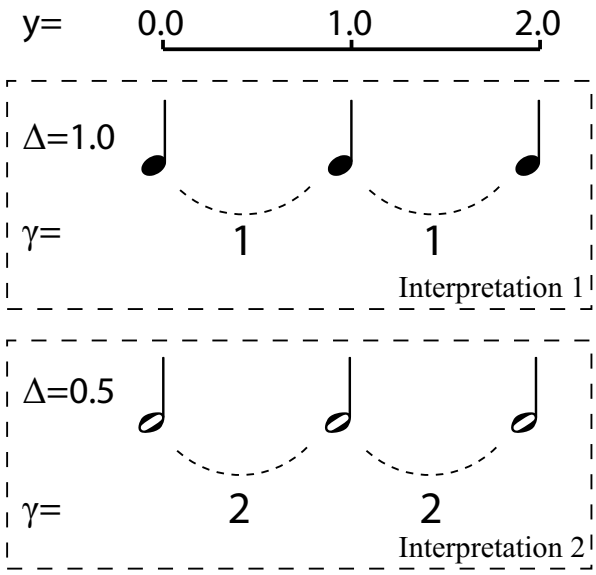

Figure 4.6: Fictive example, two interpretations of one set of onset times

In the example we will now discuss we will not include the prior probability as the likelihood turns out to be sufficient to solve the example. However, as we will see in chapter 6, when we discuss the results of performing tempo tracking in realtime, a prior probability is required to get reliable results every time a performance is played.

4.2.2 Example

In this example we will perform tempo tracking on a performance with unknown score, figure 4.7 shows a list of onset times. As the question marks indicate the score, and thus the score difference, is not known.

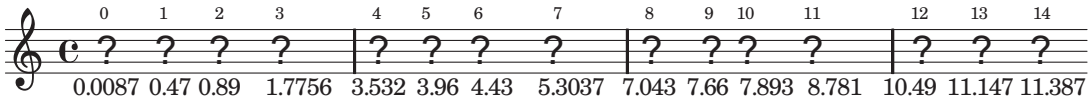

Figure 4.7: Unknown score with noisy onset time observations y, because of the unknown score the score differences are unknown too.

We will use the same parameters for the switching Kalman filter as we did in the example of the previous section. The actual calculations of the switching Kalman filter in this example are exactly the same as in the previous section, although this time the filter is used multiple times within one time step. We will therefore not go over the calculations step by step, but will simply list the final results. The values of the onset times $\tau_{k|k}$ and period $\Delta_{k|k}$ in table 4.2 are the ones with the highest likelihood after processing the final note. Additionally the score differences γ_k that were used and found best are displayed in the table.

Table 4.2: Table with various values in for example with unknown score.

k	0	1	2	3	4	5	6	7
y_k	0.0087	0.47	0.89	1.7756	3.532	3.96	4.43	5.3037
$\hat{\tau}_{k\|k}$	0.0086	0.4714	0.8978	1.7766	3.5329	3.9654	4.4163	5.3009
$\hat{\Delta}_{k\|k}$	1.0000	0.9287	0.8838	0.8806	0.8790	0.8769	0.8812	0.8825
γ_k	-	0.5	0.5	1.0	2.0	0.5	0.5	1.0

k	8	9	10	11	12	13	14
y_k	7.043	7.66	7.893	8.781	10.49	11.147	11.387
$\tau_{k\|k}$	7.0490	7.6804	7.8957	8.7732	10.4973	11.1469	11.3736
$\Delta_{k\|k}$	0.8774	0.8693	0.8687	0.8723	0.8659	0.8660	0.8691
γ_k	2.0	0.75	0.25	1.0	2.0	0.75	0.25

Next to the values in the table, a plot of the results is shown in figure 4.8. The plot includes some information that is not in the table. First of all, all the particles and their connections to their parent particles are shown and colored cyan. Because of pruning some of the routes in the plot result in a dead end, they simply turned out to be too unlikely.

The onset times and periods that achieved the highest probability (the ones that are also in the table) are connected by a dark blue line and the covariance for the points are given as green ellipses.

Finally, the dashed red line shows which particle achieved the highest likelihood at each timestep. As we can see the program first decided on a different period then it ended up choosing. However, looking closely at the values of the periods we see the route the red dashed line followed at first has a period of around 1.7, while the value it ended up choosing has a value of 0.87, which is approximately half its value. This makes perfect sense, as the period is twice as big, the score differences need to be twice as small to achieve the same onset times. The reason the algorithm decided to choose the smaller period in the end is because it ran into a sixteenth note, the score difference resolution used in this scenario was set to 0.25, as the upper line had already used this value to represent the eighth note, it did not have a lower value to represent the sixteenth note and failed to further accurately describe the onset times, this is why a switch was made to the other route.

We can check the score differences the algorithm chose by looking at the original score that was used to produce the onset times in this scenario. The original score, which was not known beforehand, was created by taking the first few notes of the song Sailing by Rod Stewart [Baker2001] and is shown in figure 4.9. When comparing the values, we see there is a perfect match. We could have even produced this score from the score difference information (except for the pitch of the notes), assuming there are no rests or overlapping notes throughout the song.

4.3 Summary

In this chapter we discussed methods for estimating the state of a linear dynamical system. We introduced the switching Kalman filter which allowed us to estimate the state of a piecewise linear model, based on a switching variable. In an example we saw how the switching Kalman filter can be used to perform tempo tracking on a musical performance of which the score was

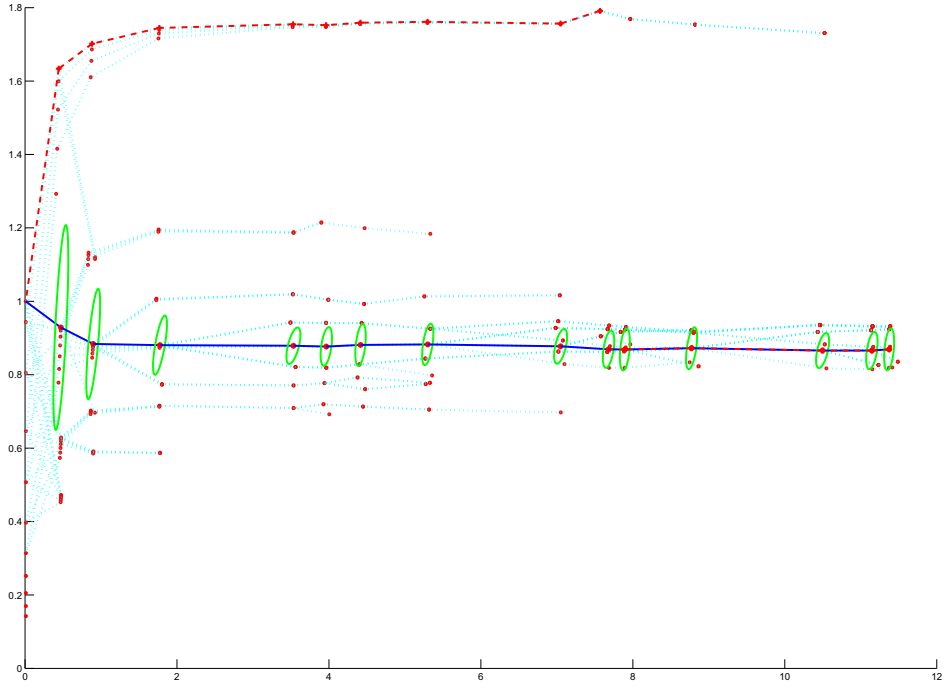

Figure 4.8: Plot of filtered onset times (x axis) and periods (y axis) together with their covariance.

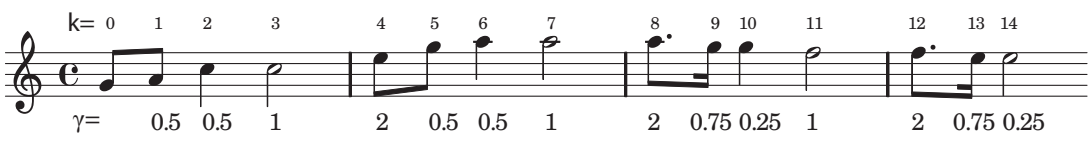

Figure 4.9: Original score used to produce onset times for scenario 3

known in advance. We then discussed the particle filter, a method for using the switching Kalman filter even when the switching variable is hidden, this allows us to perform tempo tracking on a performance with unknown score, which we discussed in the final example.

Having discussed a method that allows us to perform tempo tracking in a realtime scenario, we can implement this method in the form of a computer program, this will be the focus of our next chapter. As we want to do tempo tracking in realtime a fast implementation is required, we will discuss implementation choices and explain the architecture of the software.

Chapter 5

Implementation

With the method for doing tempo tracking explained in the previous chapter, we can now focus on creating a program to perform the actual calculations. In this chapter we will go over the design of the program and will discuss some issues encountered during implementation. A first implementation of this program was done in Matlab, as Matlab is a scripting language it allows for easy debugging, making it ideal to get a good understanding of how to implement an algorithm. The final implementation, however, was done in C++ using the win32 API to create the graphical user interface. Because we want to perform tempo tracking in realtime the speed of the program is very important, programs written in C++ produce very little overhead making the software run fast enough for realtime performance.

In section 5.1 we will explain the underlying architecture of the program and explain how this architecture allows flexibility in design and expansion of the program. Section 5.2 discusses what types of input the program can handle and how these inputs are processed. Of the realtime inputs both MIDI and raw audio data are discussed, however, only MIDI has been implemented. Then in section 5.3 we discuss some technical details regarding the implementation of the Kalman filter in C++, as the algorithm itself has been explained thoroughly we will not go into too many details in this section. Finally, in section 5.4 we discuss how we can use the tempo information, to create some interesting output. We will discuss how to generate beats and how a drum loop can be played, while somebody is playing music.

5.1 Architecture

When creating computer software, it is important to think of a good architecture for the layout of your program, a good architecture allows flexibility in altering your software as you write it. This program should process music being played and do something useful with the results, this means we can divide the program into three modules. We start with an input module which will focus on collecting the necessary data from an input device. Examples of input devices can be MIDI input, a microphone or even manual or computer generated input, which might be useful for testing the application. The data from the input device might have to be parsed for it to become useful, it can then be handed over to the processing module, which is the next module in our design. The processing module will perform the necessary calculations as discussed in the previous chapter. The processed data will be useful for our

output module, which is the final module. In the output module the data can be examined and used for some meaningful output, for example playing a clicking sound at the beat or playing a drumloop at the right tempo. This architecture is shown in a diagram in figure 5.1.

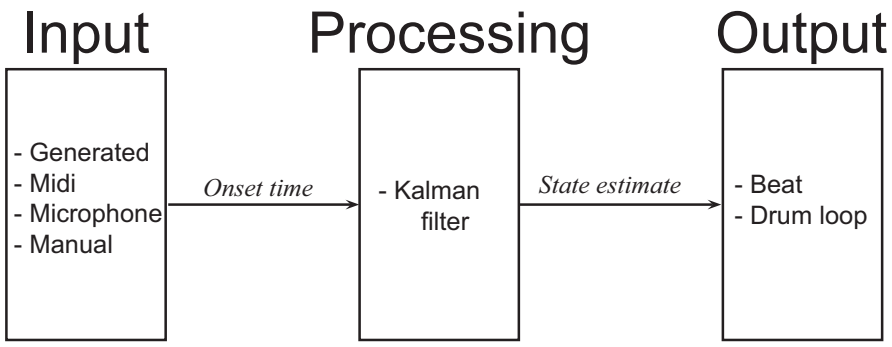

Figure 5.1: Diagram to illustrate the architecture of the tempo tracking software, italic words denote messages between various threads, while the words inside the boxes represent choices.

Each of the modules will be run in an individual thread, this will allows the program to act in time even when another thread is taking a bit longer than required. As the diagram shows, the threads communicate through messages, the nice thing about this design is that when you want to add an extra input option, you just have to make sure that it produces onset times and the rest of the program will be able to handle it. The same applies for the processing and output thread, making it very easy to add features to the program, such as trying different processing techniques or output methods. With the architecture explained we can go over each module more thoroughly.

5.2 Input

The task of the input module is to obtain the onset time and pass it to the processing module. In this section we discuss different forms of input and how we obtain the onset time for each of these forms.

5.2.1 MIDI

The term MIDI is an acronym for Musical Instrument Digital Interface, it is a communication standard that allows musical instruments and computers to talk to each other using a set of messages. These messages contain symbolic information representing the instruments action. For example, there is a note on message to indicate a certain note was played and a note off message to indicate it stopped. Unfortunately most instruments do not come with a MIDI connector, as most instruments are not electric, but for almost every instrument there is either a MIDI equivalent or MIDI kit available allowing it to produce MIDI messages [Guerin2002].

To allow the program to handle MIDI messages the PortMIDI library was used, which is a cross-platform, open source library available at the Carnegie Mellon website. The library is able to read from and write to a MIDI port, making it possible to use the library for receiving MIDI messages as well as MIDI playback. Because each MIDI message is accompanied by a

timestamp, we only need to filter for note on messages in order to get the onset time. After passing this onset time to the processing module the task for the input module is done.

5.2.2 Microphone (not implemented)

Although not implemented, it is also possible to use a microphone as input device. Because the data from a microphone is raw audio data, we will need to further process this data to obtain the onset times. The raw audio data can be obtained by using the PortAudio library, this is the raw audio equivalent of the PortMIDI library. We can then apply a wavelet analysis on the data using the wave++ library, which is a C++ library for performing wavelet analysis, time-frequency analysis and Fourier analysis. For monophonic music it might suffice to detect frequency changes and use that as a basis for finding onset times, however, as this is quite a complicated task this has not been implemented and has therefore not been tested. An overview of onset detection techniques in polyphonic music can be found in Bello (2003).

5.2.3 Generated and Manual

Because tempo tracking is a task still being researched, it could be useful to try certain scenarios to see how the algorithm performs. It is for this reason that both computer generated input and manual input have been added to the program. Computer generated input generates noisy onset times according to a number of parameters such as the tempo and a noise covariance matrix. The manual input option can be used to enter onset times by hand, this can be handy to try a certain scenario over and over, using different parameters for the processing step.

5.3 Processing

The task of the processing module is to take the incoming noisy onset time and produce an estimate of the actual onset time and period. Although all sorts of processing techniques can be used, only the particle filter has been implemented. We have discussed the details of both the particle filter and the switching Kalman filter in the previous chapter, so we will not go over too much programming details here. The implementation of the matrix calculations, needed for using the switching Kalman filter, have been written out to a scalar level for a two element state vector. This severely limits the reusability of the switching Kalman filter code, but allows for a very simple and fast implementation. As we are going to process realtime data this approach was chosen. The calculations of the particle filter are pretty straight forward, a C++ standard template library multimap was used to store the particles at each timestep, this way the particles are automatically sorted by likelihood making it very easy to use the n-best results. Next to the state estimate with the highest likelihood, the score position is also sent to the output thread, as we will see in the next section this information is required for getting some interesting output.

5.4 Output

In the output thread the tempo information obtained from the processing module is used to produce interesting output. Timing is very important when it comes to output, so to make sure everything happens on time the output thread uses two functions. One function

continuously monitors the clock to see if it is time to output an event, once an event has been played it calculates the timestamp for a new event and prepares it. Another function handles the incoming data from the processing module, updating any timestamps based on this data. This structure guarantees on the one hand that output is always played on time, while on the other hand guarantees that the most recent results have been used to get as accurate timing information as possible.

5.4.1 Beat

The beat is a pulse that occurs on the basic time unit in music, when one taps along his foot each tap is a beat. The moment a beat is played depends purely on the tempo, a fast tempo means beats are played at a fast pace too. In outputting the beat it is important to constantly incorporate the tempo made available by the processing module to maintain accurate beat times. Generating a beat while playing music gives a good indication whether the tempo tracking was performed successfully, an incorrectly played beat is very easy to notice.

The processing module provides the output module with the onset time, the period and the score position of the processed note. To find the timestamp for a beat we simply need to round the score position to the nearest highest integer value, this gives us the score position of the beat. We can then calculate the score difference between the note and the beat and use that value to calculate the timestamp of the beat. These calculations are performed every time the processing thread provides the output module with new tempo information. Another function continuously checks whether the calculated beat time has been reached and outputs the beat if so, it then calculates the timestamp of the next beat. This way a beat is always played, using the latest tempo information available.

5.4.2 Drum loop

A drum loop is a sequence of drum MIDI events that can be played in a repeating fashion to simulate a background drummer. The time at which each of the individual drum events are played depends on the tempo of the music. To calculate these times we first need to load the drum events from a MIDI file and determine the score position of each individual drum event, this can be obtained from the information in the file. With the tempo information obtained from the processing module, together with the score position of every drum event, we can calculate the timestamp of each event in a similar matter as with the calculation of the beat time. We can then use an index to indicate which drum event needs to be played next and reset the index at the end of the sequence to create the looping effect. The same structure is used as with the beats, in that the drum loop in continuously played and timestamps accurately updated using the latest tempo information available.

5.5 Summary

In this chapter we discussed how tempo tracking can be implemented in a computer program, we used an approach in which the program is divided into three modules, an input, processing and output module. Each of these modules can be used in different ways, allowing us to try different configurations and giving us a tool to perform tempo tracking in all sorts of scenarios. Using this tool we can perform various tests to determine the accuracy, speed and ability to

recover from errors of the implementation, in the next chapter we will introduce various testing methods and discuss the results produced from these tests.

Chapter 6

Results

Having created a tool to perform tempo tracking we can now determine if our implementation qualifies for a realtime scenario. In section 6.1 we will use a dataset, used more often in tempo tracking, to test the accuracy of our method. We will then test the speed of the application in section 6.2, comparing an implementation in C++ with one in Matlab. Finally, we will discuss how the program performs in a realtime scenario in section 6.3, learning how various parameters affect the behavior of the program..

6.1 Accuracy

One of the most important results in testing an algorithm is its accuracy, without a high accuracy things like speed become less and less relevant as nobody needs a fast algorithm that completely messes up a task. In this section we discuss how we test the accuracy of our program and compare the results with another implementation.

The model we used is taken from Cemgil, in his thesis he also discusses a number of tests performed on his model, however, his tests are done only offline [Cemgil2004]. In offline processing all observations are available beforehand, therefore one can determine which interpretations fits best taking all observations into consideration, additionally Kalman smoothing can be performed to get even better results. The difference in our approach is that we are dealing with a realtime implementation of the model. This means that events that have already been processed can no longer be altered, therefore we expect the accuracy to be somewhat lower than in the offline case.

The data set we will use to test the accuracy is the beatles data set, this is the same data set that was used to test the offline implementation. The beatles data set consists of MIDI recorded piano performances by twelve different pianists playing Michelle and Yesterday by the Beatles. Of the twelve pianists there are four professional jazz players, five professional classical players and three amateur players. The performances of the pieces were done in three different tempo conditions, namely slow, normal and fast, it was left to the players to decide how to interpret these different tempos. For each tempo condition a total of three repetitions were recorded, all this results in a total of 216 performances (= 12 pianists $\times$ 3 tempi $\times$ 3 repetitions $\times$ 2 pieces). The data set can be found at `http://www.nici.kun.nl/mmm/` under download and then data archive.

Given the true score differences $\gamma_{1:K}^{true}$ we are able to measure the quality of a solution

by calculating the edit distance $e(\gamma_{1:K})$. The edit distance compares the true score differences with the score differences found by the algorithm and counts the number of incorrectly identified notes. In formula we write

$$e(\gamma_{1:K}) = \sum_{k=1}^{K}(1 - \delta(\gamma_k - \gamma_k^{true}))$$

where the δ function is a boolean function giving 1 when the score differences are equal and 0 when not [Cemgil2004].

The percentage of misclassified notes is shown in table 6.1, for both the model as used in Cemgil and our realtime implementation. The tests have been performed keeping the n-best particles, for various values of n.

Table 6.1: Table showing the percentage of edit distance (lower is better), tests were performed using various values for the n-best particles. Both the results of Cemgils offline tests and our realtime tests are included.

n	1	10	20	50
Cemgil	8.8	7	5.2	5.1
Realtime	19.48	17.15	17.00	16.98

These results show that the offline implementation (referred to as Cemgil), does the task a lot better than the realtime implementation. This was expected as the offline implementation is able to take all observations into account and perform Kalman smoothing.

The results for the realtime implementation might seem rather high, making it questionable whether the realtime implementation is accurate enough. However, from studying the results it appears that most of the errors are clustered together in a sequence. This means that the algorithm has trouble with a certain part of the piano performance, but is able to pick up the right track afterwards. In a realtime performance, the program will give incorrect output for a while and then continue giving the right output, which is a lot less frustrating than continuously giving a few incorrect outputs throughout the entire performance.

To determine whether the accuracy lives up to our requirements we will simply have to try the program in realtime ourselves, the experiences of these trials are discussed in a later section of this chapter.

6.2 Speed

Because we want to perform tempo tracking in realtime it is important that the application is capable of performing the necessary calculations in a reasonable timespan. In this section we will discuss a test done to measure the speed of the application and determine whether it will be fast enough to perform tempo tracking in realtime. In the previous chapter we discussed the implementation of the program, and argued that an implementation in C++ would be required for realtime processing as an implementation in Matlab would produce too slow results. Because the algorithm has been implemented both in C++ and in Matlab we are able to test this theory by measuring the calculation time of both implementations. Our algorithm uses a particle filter which uses the n-best results in each iteration, the higher the value of n the more particles will be used and thus the more calculations are required.

For this test we used one of the performances from the beatles data set, the performance consists of 293 MIDI events meaning the particle filter will go over 293 iterations. To clearly show the difference in calculation times we tested both implementations for various values of n, table 6.2 shows the time in seconds both implementations needed to process the entire file.

Table 6.2: Time in seconds the algorithm needs keeping n particles at each iteration.

n	1	10	25	50	100	250	500	1000
C++	2.9 s	3.0 s	3.2 s	3.4 s	3.4 s	4.4 s	6.3 s	12.4 s
Matlab	4.7 s	25.1 s	60.6 s	113.8 s	229.5 s	702.3 s	-	-

As the values in the table show, the C++ implementation is a lot faster than the Matlab implementation, the time for Matlab to process the file with $n = 500$ and $n = 1000$ have not even been included because Matlab crashed after crunching numbers for over 15 minutes. Looking at the times for the C++ implementation, we see that for $n = 100$ and lower, the program needs around 3 seconds to process the entire file. As the file consists of 293 MIDI events it takes an average of around 10 milliseconds for the program to process one event. This means the program will be able to generate output based on information that is 10 milliseconds old. As it seems unlikely to have very strong changes in tempo in such a short timespan, the program should be fast enough for realtime processing. Also, with a processing time of 10 milliseconds per event, the program will be capable of processing 100 events in one second. This seems more than enough for any type of performance a human being can give.

6.3 Realtime Experience

Although the accuracy is pretty good and the program is rather fast at processing input, we will not know how well it does in realtime until we actually try it. In this section we will explain how the program reacts to various performances played on a MIDI keyboard and discuss how different parameters affect the programs response.

We have already discussed including prior probabilities to differentiate between different interpretations, it turns out these prior probabilities are very important in getting the results we are after. In various attempts playing the piano without the use of prior probabilities, the program appears to have a mind of its own and plays beats at times that do not match the tapping of the performer at all. Listening more carefully, the speed at which the beats are played does increase with the increasing speed the performer plays at, they just do not match the performers tapping. When analyzing the data afterwards, it turns out the algorithm simply interprets some notes incorrectly making it shift the entire performance, as a result the algorithm continues to have a shifted beat track. This is shown graphically in figure 6.1, the values above the line are the score positions c and the values below the score differences γ. As the example shows even though only two score differences in the filtered result are different from the actual values, the rest of the track suffers from this error as everything gets shifted in time. The figure also shows how the beat (indicated by the arrows) gets shifted as a result of this, making the program respond incorrectly to the performance.

This example shows that, because the program continues to select the right score difference for the incoming onset times, it has no reason to correct the error made in the past, leaving the score position shifted. We can avoid this kind of shifting behaviour by introducing a prior probability saying we prefer integer score positions, meaning we would rather have

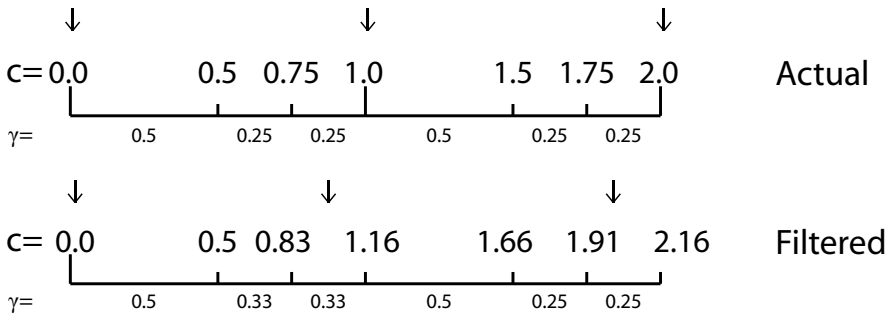

Figure 6.1: Fictive example, the filtered score positions $c_{filtered}$ are shifted compared to the actual score positions c_{actual}, therefore the beat (indicated by the arrow) is shifted as well and played at the wrong moment in time.

an interpretation in which some notes coincide with a beat. Using such a prior will cause a scenario with the shifting behaviour to become less and less attractive as more and more notes fail to coincide with the beat. Because we are working with a particle filter, we continue to track certain alternative solutions, eventually the algorithm will switch to a different solution, correcting the error made in the past. Because we are working in realtime the effect of the error will already have been outputted, however, for the newly processed notes the output will now be as expected.

By providing the program with more information through the use of prior probabilities, we are able to make it react more precisely to our needs. However, by incorporating more information about a performance being played, we are also restricting the programs flexibility. The problem here is that it is difficult to find prior probabilities that apply to all possible musical performances. Because the range of music genres is so wide, almost every assumption made about the music in advance rules out a certain other genre. One solution could be to have the user select a certain genre in advance, so that the program will know which set of prior probabilities to use. This would make the program very accurate, while maintaining its flexibility.

Next to the prior probabilities we can also use different values for various parameters of the Kalman filter. Two parameters that affect the behaviour of the algorithm quite strongly are the process noise covariance Q and the measurement noise covariance R. The process noise covariance expresses our belief in the results of the model, while the measurement noise covariance expresses our belief in the observations. As our model assumes a constant tempo the process noise should not be too low, to allow the algorithm to incorporate tempo fluctuations. On the other hand the measurement noise should not be too low either, as a human performer is not able to play with a very high accuracy on a millisecond level.

When using the beat as output, it gives the best results when using a process noise covariance that is not too low. This way the tempo tracker is able to track every fluctuation in tempo the performer makes, making the beat coincide perfectly with the notes. However, using the same value for the covariance when playing the drumloop, the effect is somewhat unrealistic. Because a drummer normally produces a rhythm in a rather constant tempo, it sounds rather strange to hear a drum loop change tempo slightly at every beat. By choosing a process noise covariance that is somewhat lower, we request a more constant tempo. This difference can be seen clearly in figures 6.2 and 6.3, where the tempo curve is plotted using the

high and low covariance, respectively. We see that the tempo curve using the lower covariance gives a much smoother result, which is preferable for a drum loop.

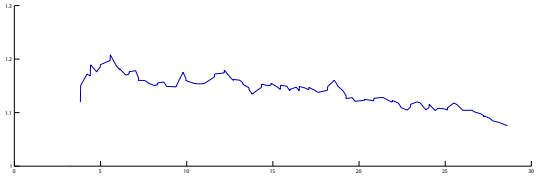

Figure 6.2: Tempo curve using high covariance, making it very sensitive to change (x-axis: time, y-axis: period)

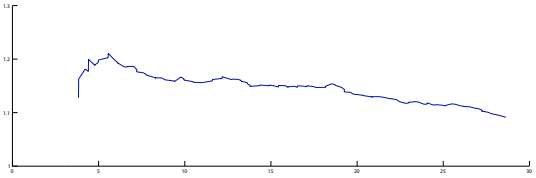

Figure 6.3: Tempo curve using low covariance, making it less sensitive to change (x-axis: time, y-axis: period)

The measure noise covariance appears to be less flexible in assigning a value. Choosing a value that is too low makes the algorithm very strict in judging the notes that pass by. When only a single note is played with a tiny delay, either because of error or rhythmic effect, the algorithm switches to an alternative interpretation. When choosing a value that is too high, it takes too long for the algorithm to incorporate the effect of a tempo change in the performance. This makes it chase the performance in a delayed fashion all of the time.

Overall, when knowing how to properly use the parameters of the program, it gives very good results in giving a realtime response to a performance. Setting the proper priors to match a specific genre and choosing the right covariances for the selected output makes the program a fun and useful tool for musical experiments.

6.4 Summary

In this section we measured the accuracy and calculation time of our implementation to determine whether is qualifies for realtime processing. By comparing the accuracy with the offline implementation, we get a good impression of its performance. And by comparing the calculation time with the implementation in Matlab, the choice for C++ as a programming language seems justified. The program responds very well to performances played, given that the right settings are chosen.

Chapter 7

Conclusion

In developing a program for realtime tempo tracking, a number of criteria are important; the program has to by quick at processing input, must give a good accuracy and must be able to recover from errors. In this thesis we first explained a model for tempo tracking proposed by Cemgil, we then learned how we can use the particle filter to find the most probable interpretation of a given performance [Cemgil2004]. After that we discussed the implementation of the particle filter in C++ and were able to perform various tests using this implementation. As a result the program turns out to be very fast at processing input, give a good accuracy and has no trouble recovering from errors. It therefore fits the criteria required for a realtime implementation very well, giving us a powerful tool for doing tempo tracking.

The tempo tracking approach used in this thesis is a very flexible one, meaning it can be used for practically any type of musical setting as long as the right parameters (such as prior probabilities, set of score difference and Kalman settings) are used. In choosing these parameters it is important not to restrict the program to one type of performance, making it necessary to alter it every time a different type of music needs to be processed. The trick here is to use those parameters that apply to a broad class of musical genres, however, it turns out to be rather difficult to find those parameters. For example, in using prior probabilities it is difficult to find a set of priors that applies to every type of music. A study into what features different musical genres share, would allow us to express preference for such features. This would make the program more accurate while maintaining its flexibility.

With respect to the settings of the Kalman filter (such as the values of the noise covariances) a learning algorithm could be used to find the optimal values. A large collection of performances from different musical genres could be used as a training set. If it turns out that no single set of values can be used for all possible performances, a collection of sets could be formed and included in the particle filter as hypotheses. That way the particle filter can figure out which set works best for a particular performance.

Next to finding the ideal parameters, some improvements to the model can also be considered. The model currently only takes timing information into account to base its decision on, however, things like pitch, chords, the duration of notes and the relative amplitude have shown to improve results in other studies. Furthermore, the model currently treats each onset time as equally relevant to the task of tempo tracking. But when processing multiple instruments, the percussion instruments are likely to provide stronger cues regarding the tempo

than for example a flute [Cemgil2004].

Finally, to allow the program to process every type of musical instrument, the processing of raw audio data should be incorporated. Although almost every instrument has a MIDI capable equivalent, it is preferable to use a microphone for recording a musical performance. This takes away an important restriction, because it would allow a performer to use his own instrument. Next to using a microphone, the processing of audio files such as mp3 or wav could also be implemented. Of course, in order to process raw audio data a pre-processing step should be performed to extract the onset times from the performance. Because the algorithm strongly depends on those onset times, extraction of them should be done with great accuracy, making it a very difficult and challenging task.

All these improvements should lead to a very accurate and widely applicable tempo tracker, making it perfectly suitable for use in applications such as an Interactive Music Performance System.

Appendix A

Kalman filter derivation

The derivation in this section was established by combining parts from [Brown1992] and [Maybeck1979]. We start by stating the model for a linear dynamical system. In general form this model is

$$x_{k+1} = A_k x_k + w_k \tag{A.1}$$

The observation of the process is given as

$$y_k = C_k x_k + v_k \tag{A.2}$$

For clarities sake we will go over each of these elements one more time:

x_k = $(n \times 1)$ process state vector

A_k = $(n \times n)$ transition matrix, relating x_k to x_{k+1}

w_k = $(n \times 1)$ system noise vector, assumed to be white and Gaussian

Q_k = $(n \times n)$ covariance matrix for w_k

y_k = $(m \times 1)$ measurement vector

C_k = $(m \times n)$ observation matrix, giving the noiseless connection between the measurement and the state vector

v_k = $(m \times 1)$ measurement error, assumed to have zero crosscorrelation with w_k, white and Gaussian

R_k = $(m \times m)$ covariance matrix for v_k

The goal in using the Kalman filter is to estimate the next state of a system, given the previous state. In doing so it should filter out as much noise as possible while incorporating the observation made. The state of the filter is represented by two variables, the state estimate $\hat{x}$ and the corresponding error covariance matrix P. As can be seen from this notation, we will use a hat to indicate we are dealing with an estimate of the state.

The Kalman filter is a set of equations that can be summarized into two steps. The first step is called the *prediction step*, which predicts the value of the new state that the system will arrive in. The second step is called the *correction step*, as it corrects the prediction that was made by incorporating information gained from an observation. Each of these steps gives us a state estimate, however, the different steps produce different versions of the state.

The prediction step gives us the a priori state estimate as it incorporates all knowledge before or prior to the current point k, we will denote the variables in this state as $\hat{x}_{k|k-1}$ and $P_{k|k-1}$. This notation indicates we are dealing with variable $\hat{x}$ at time k given the state at $k-1$, as the Kalman filter is a recursive filter the state at $k-1$ incorporates the information of the states before that point.

The correction step gives us the posteriori state as it is the state after the observation has been made. Therefore, the posteriori state incorporates all knowledge up to the current point k, and is denoted as $\hat{x}_{k|k}$ and $P_{k|k}$. The a priori estimate is our best estimate of $\hat{x}_k$ prior to incorporating the observation at point k. The posteriori state is the optimal estimate that the Kalman filter produces.

We assume that we have an initial estimate $\hat{x}_{k|k-1}$ and covariance $P_{k|k-1}$ of the process at some point in time k and that this estimate is based on all our knowledge about the process prior to point k. This initial estimate is an a priori estimate which means that the iterations of the Kalman filter will start at the correction step.

Having explained the notation that we will use for the filter, we will now proceed to the derivation. Because in the prediction step we calculate the prediction that our model gives, we can derive the equations for this step by taking the expected value and covariance of our model equations.

A.1 Prediction step

Model Equation:

$$x_{k+1} = Ax_k + w_k \tag{A.3}$$

Expected value:

$$\langle x_{k+1} \rangle = \langle Ax_k + w_k \rangle \tag{A.4}$$
$$= A\langle x_k \rangle + \langle w_k \rangle \tag{A.5}$$

Because the mean of the error w_k is zero, the expected value is zero too:

$$\langle x_{k+1} \rangle = A\langle x_k \rangle + 0 \tag{A.6}$$
$$= A\langle x_k \rangle \tag{A.7}$$

Because $\langle x \rangle = \hat{x}$ the formula for the prediction step becomes:

$$\mathbf{\hat{x}_{k+1|k}} = \mathbf{A\hat{x}_{k|k}} \tag{A.8}$$

Note that this is the a priori state, which can be seen by the subscript of $\hat{x}$, this a priori state vector gives us the predicted value of the state before any observation was made. We shall now proceed to calculating the a priori error covariance matrix $P_{k|k-1}$. The covariance is calculated by $P = \langle x^2 \rangle - \langle x \rangle^2$, we will therefore first simplify these individual terms to

make the derivation more readable. To improve readability even further, we will first state formulas with their subscript, but will hide the subscript for the remainder of the derivation.

Because $\langle x^2 \rangle \neq \langle x \rangle^2$ we have to calculate $\langle x^2 \rangle$ separately, first we calculate x^2:

$$x_{k+1}.x_{k+1}^T = (Ax_k + w_k)(x_k^T A^T + w_k^T) \tag{A.9}$$
$$= Axx^T A^T + wx^T A^T + Axw^T + ww^T \tag{A.10}$$

We can now drop two terms as the expected value of w is zero:

$$\langle x_{k+1}.x_{k+1}^T \rangle = \langle Ax_k x_k^T A^T + w_k x_k^T A^T + Ax_k w_k^T + w_k w_k^T \rangle \tag{A.11}$$
$$= A\langle xx^T \rangle A^T + \langle w \rangle \langle x^T \rangle A^T + A\langle x \rangle \langle w^T \rangle + \langle ww^T \rangle \tag{A.12}$$
$$= A\langle xx^T \rangle A^T + 0\langle x^T \rangle A^T + A\langle x \rangle 0 + \langle ww^T \rangle \tag{A.13}$$
$$= A\langle xx^T \rangle A^T + \langle ww^T \rangle \tag{A.14}$$

To further simplify this formula we first have to calculate the covariance of w_k:

$$cov(w_k) = \langle w_k^2 \rangle - \langle w_k \rangle^2 \tag{A.15}$$
$$= \langle ww^T \rangle - 0 \tag{A.16}$$
$$= \langle ww^T \rangle \tag{A.17}$$

We see that the term $\langle ww^T \rangle$ is equivalent to the covariance of the error term w_k which is Q, therefore we get:

$$\langle x_{k+1}.x_{k+1}^T \rangle = A\langle x_k x_k^T \rangle A^T + \langle ww^T \rangle \tag{A.18}$$
$$= A\langle x_k x_k^T \rangle A^T + Q \tag{A.19}$$
$$\tag{A.20}$$

Now we proceed to calculating the covariance:

$$P_{k+1} = \langle x_{k+1}^2 \rangle - \langle x_{k+1} \rangle^2 \tag{A.21}$$
$$= \langle x_{k+1}.x_{k+1}^T \rangle - \langle x_{k+1} \rangle \langle x_{k+1} \rangle^T \tag{A.22}$$
$$= A\langle x_k x_k^T \rangle A^T + Q - A\langle x_k \rangle \langle x_k \rangle^T A^T \tag{A.23}$$

We can take the two equations surrounded by the A matrices together to arrive at the final form:

$$P_{k+1} = A(\langle x_k x_k^T \rangle - \langle x_k \rangle \langle x_k \rangle^T) A^T + Q \tag{A.24}$$

As the term in between the brackets is equivalent to the covariance of the previous state our formula for the a priori error covariance is:

$$\mathbf{P_{k+1|k}} = \mathbf{A P_{k|k} A^T} + \mathbf{Q} \tag{A.25}$$

A.2 Correction step

With the formulas for the a priori estimate $\hat{x}_{k+1|k}$ and covariance $P_{k+1|k}$ we have all the formulas of the prediction step. Now we want to proceed to the correction step in which we use the measurement y_k to improve the a priori estimate. This is done by a linear blending of the noisy measurement and the a priori estimate.

$$\hat{\mathbf{x}}_{\mathbf{k|k}} = \hat{\mathbf{x}}_{\mathbf{k|k-1}} + \mathbf{K_k}(\mathbf{y_k} - \mathbf{C}\hat{\mathbf{x}}_{\mathbf{k|k-1}}) \tag{A.26}$$

where C is the observation matrix as used in formula A.2 and K_k is the blending factor, that is yet to be determined. The difference $(y_k - Cx_{k|k-1})$ in this function is called the residual. The residual reflects the discrepancy between the predicted estimate $Cx_{k|k-1}$ and the measurement y_k. A residual of zero means that the two are in complete agreement. The blending factor K_k determines to what degree the residual should be used to update the a priori estimate $x_{k|k-1}$. So our task now is to find that value of K_k that yields an optimal updated estimate. We will derive the formula of K_k by first deriving the formula for $P_{k|k}$.

Calculate covariance according to $P = \langle (x - \hat{x})(x - \hat{x}) \rangle$, for readability we will first write this as $P = cov(x - \hat{x})$

$$P_{k|k} = cov(x_k - \hat{x}_{k|k}) \tag{A.27}$$

We now substitute $\hat{x}_{k|k}$ with the formulas we have obtained so far and work out the brackets.

$$\begin{aligned}
P_{k|k} &= cov(x_k - (\hat{x}_{k|k-1} + K_k(y_k - C\hat{x}_{k|k-1}))) && \text{(A.28)}\\
&= cov(x_k - (\hat{x}_{k|k-1} + K_k(Cx_k + v_k - C\hat{x}_{k|k-1}))) && \text{(A.29)}\\
&= cov(x_k - (\hat{x}_{k|k-1} + K_kCx_k + K_kv_k - K_kC\hat{x}_{k|k-1})) && \text{(A.30)}\\
&= cov(x_k - \hat{x}_{k|k-1} - K_kCx_k - K_kv_k + K_kC\hat{x}_{k|k-1}) && \text{(A.31)}
\end{aligned}$$

Now we group the $x_k - \hat{x}_{k|k-1}$ terms together.

$$P_{k|k} = cov((I - K_kC)(x_k - \hat{x}_{k|k-1}) - K_kv_k) \tag{A.32}$$

As v_k is not correlated with the rest, we can move it outside the brackets

$$P_{k|k} = cov((I - K_kC)(x_k - \hat{x}_{k|k-1})) + cov(K_kv_k) \tag{A.33}$$

We can now move the constants outside the covariance terms as well

$$P_{k|k} = (I - K_kC)cov(x_k - \hat{x}_{k|k-1})(I - K_kC)^T + K_kcov(v_k)K_k^T \tag{A.34}$$

Now we see we actually ended up with the covariance $P_{k|k-1}$ and the covariance of v_k which is R.

$$P_{k|k} = (I - K_kC)P_{k|k-1}(I - K_kC)^T + K_kRK_k^T \tag{A.35}$$

This formula is valid for every value of K_k (also non optimal ones). However, we wish to find that value for K_k that optimizes the formula. We will see later that once we have found

the formula to calculate the optimal value of K_k, we can further simplify our function for $P_{k|k}$. What we mean when we say we are looking for the value of K_k that optimizes $P_{k|k}$, is that we are looking for that particular K_k that minimizes the individual terms along the major diagonal of $P_{k|k}$, as these terms represent the estimation error variances for the elements of the state vector being estimated. Therefore this minimization is equivalent to minimizing the trace of $P_{k|k}$. Because the trace of a matrix is defined the sum of the elements on the main diagonal (the diagonal from the upper left to the lower right). So taking again the formula of $P_{k|k}$ we ended up so far and working out the brackets.

$$P_{k|k} = (I - K_k C)P_{k|k-1}(I - K_k C)^T + K_k R K_k^T \tag{A.36}$$
$$= (P_{k|k-1} - K_k C P_{k|k-1})(I - K_k C)^T + K_k R K_k^T \tag{A.37}$$
$$= P_{k|k-1} - K_k C P_{k|k-1} - P_{k|k-1} C^T K_k^T + K_k C P_{k|k-1} C^T K_k^T + K_k R K_k^T \tag{A.38}$$

Now we group the two terms surrounded by K_k and introduce a temporary variable for that term

$$P_{k|k} = P_{k|k-1} - K_k C P_{k|k-1} - P_{k|k-1} C^T K_k^T + K_k (C P_{k|k-1} C^T + R) K_k^T \tag{A.39}$$
$$= P_{k|k-1} - K_k C P_{k|k-1} - P_{k|k-1} C^T K_k^T + K_k S K_k^T \tag{A.40}$$

Now we have worked out the formula to such a degree that we can take the derivative of the trace and equal it to zero. We will first introduce some standard formulas for derivatives of traces taken from [Roweis1999] and [Brown1992].

$$\frac{\partial Tr[X^T A]}{\partial X} = \frac{\partial Tr[A X^T]}{\partial X} = A^T \tag{A.41}$$
$$\frac{\partial Tr[X A X^T]}{\partial X} = 2XA \tag{A.42}$$

Now we apply these rules to our formula and equal it to zero.

$$\frac{\partial Tr[P_{k|k}]}{\partial K_k} = -2(C P_{k|k-1})^T + 2K_k S = 0 \tag{A.43}$$

We can now solve this formula for K_k.

$$2K_k S = 2(C P_{k|k-1})^T \tag{A.44}$$
$$K_k S = (C P_{k|k-1})^T \tag{A.45}$$
$$K_k S = P_{k|k-1} C^T \tag{A.46}$$
$$K_k = P_{k|k-1} C^T S^{-1} \tag{A.47}$$

Substituting the value for S we get the final form for K_k known as the optimal Kalman gain.

$$\mathbf{K_k = P_{k|k-1} C^T (C P_{k|k-1} C^T + R)^{-1}} \tag{A.48}$$

Now that we have the formula for the optimal value of K_k we can return to our formula for calculating the covariance $P_{k|k}$. Remember, the formula we had derived for $P_{k|k}$ so far

was valid for every value of K_k. We shall now use the formula for the optimal value of K_k to further simplify the formula of $P_{k|k}$. We start by restating our optimal formula for K_k using the substituted variable S.

$$K_k = P_{k|k-1}C^T S^{-1} \tag{A.49}$$

Now multiplying both sides with SK_k^T we get:

$$K_k S K_k^T = P_{k|k-1}C^T K_k^T \tag{A.50}$$

Now we continue with the formula for $P_{k|k}$ we had so far, we can substitute the formula for the kalman gain in the covariance formula.

$$\begin{aligned} P_{k|k} &= P_{k|k-1} - K_k C P_{k|k-1} - P_{k|k-1}C^T K_k^T + K_k S K_k^T && \text{(A.51)} \\ &= P_{k|k-1} - K_k C P_{k|k-1} - P_{k|k-1}C^T K_k^T + P_{k|k-1}C^T K_k^T && \text{(A.52)} \\ &= P_{k|k-1} - K_k C P_{k|k-1} && \text{(A.53)} \end{aligned}$$

Bringing the a priori covariance outside the brackets we get the final form for the formula.

$$\mathbf{P_{k|k} = (I - K_k C) P_{k|k-1}} \tag{A.54}$$

A.3 Summary

To summarize the algorithm we state all the formulas for the Kalman filter one last time.

Prediction Step:

$$\hat{x}_{k+1|k} = A\hat{x}_{k|k} \tag{A.55}$$

$$P_{k+1|k} = AP_{k|k}A^T + Q \tag{A.56}$$

Correction Step:

$$K_k = P_{k|k-1}C^T(CP_{k|k-1}C^T + R)^{-1} \tag{A.57}$$

$$\hat{x}_{k|k} = \hat{x}_{k|k-1} + K_k(y_k - C\hat{x}_{k|k-1}) \tag{A.58}$$

$$P_{k|k} = (I - K_k C)P_{k|k-1} \tag{A.59}$$

Bibliography

[Alghoniemy1999] M. Alghoniemy and A. Tewfik. Rhythm and periodicity detection in polyphonic music. *Proceedings of the 1999 IEEE Workshop on Multimedia Signal Processing*, pages 185–190, 1999.

[Baker2001] Kenneth Baker. *The Afternoon Pianist*. Wise publications, 2001.

[Bertsekas2002] Dimitri P. Bertsekas and John N. Tsitsiklis. *Introduction to Probability*. Athena Scientific, 2002.

[Bretscher2001] Otto Bretscher. *Linear Algebra with Applications*. Prentice Hall, second edition, 2001.

[Brown1992] R.G. Brown and P.Y.C. Hwang. *Introduction to Random Signals and Applied Kalman filtering*. John Wiley & Sons, Inc, second edition, 1992.

[Cemgil2004] Ali Taylan Cemgil. *Bayesian Music Transcription*. PhD thesis, Radboud University of Nijmegen, 2004.

[Desain1999] P. Desain and H. Honing. Computational models of beat induction: The rule-based approach. *Journal of New Music Research*, 28:29–42, 1999.

[Dixon2000] S. Dixon and E. Cambouropoulos. Beat tracking with musical knowledge. *Proceedings of the 2000 European Conference on Artificial Intelligence, Berlin*, pages 626–630, 2000.

[Dixon2001] S. Dixon. Automatic extraction of tempo and beat from expressive performances. *Journal of New Music Research*, 30:39–58, 2001.

[Foote2001] J. Foote and S. Uchihashi. The beat spectrum: A new approach to rhythm analysis. *Proceedings of the 2001 international Conference on Multimedia and Expo, New York*, pages 881–884, 2001.

[Gasser1999] D. Eck M. Gasser and R. Port. Meter as mechanism: A neural network model that learns metrical patterns. *Connection Science*, 11:187–216, 1999.

[Goto1999] M. Goto and Y. Muraoka. Real-time beat tracking for drumless audio signals: Chord change detection for musical decisions. *Speech Communication*, 27:311–335, 1999.

[Gouyon2005] F. Gouyon and S. Dixon. A review of automatic rhythm description systems. *Computer Music Journal*, 29:34–54, 2005.

[Guerin2002] Robert Guerin. *MIDI Power*. Course Technology PTR, 2002.

[Jordan2005] Michael I. Jordan. An introduction to graphical models. To be published, 2005.

[Kalman1960] R.E. Kalman. A new approach to linear filtering and prediction problems. *Transactions of the ASME–Journal of Basic Engineering*, 82(Series D):35–45, 1960.

[Large1994] E. Large and E. Kolen. Resonance and the perception of musical meter. *Connection Science*, 6:177–208, 1994.

[Longuet-Higgins1982] C. Longuet-Higgins and C. Lee. Perception of musical rhythms. *Perception*, 11:115–128, 1982.

[Maybeck1979] Peter S. Maybeck. *Stochastic Models, Estimation, and Control*, volume 141 of *Mathematics in Science and Engineering*. Academic Press, 1979.

[McAuley1995] J. McAuley. *Perception of Time as Phase: Towards and Adaptive-Oscillator Model of Rhythmic Pattern Processing*. PhD thesis, Indiana University, Bloomington, 1995.

[Mitchell1997] Tom M. Mitchell. *Machine Learning*. McGraw-Hill, 1997.

[Murphy1998] Kevin P. Murphy. *Switching Kalman Filters*, 1998.

[MurphyWeb1998] Kevin P. Murphy. `http://www.cs.ubc.ca/~murphyk/Bayes/bnintro.html`, 1998.

[Rosenthal1992] D. Rosenthal. *Machine Rhythm: Computer Emulation of Human Rhythm Perception*. PhD thesis, Massachusetts Institute of Technology, 1992.

[Roweis1999] Sam Roweis. *Matrix Identities*, 1999.

[Smith1996] L. Smith. Modeling rhythm perception by continuous time-frequency analysis. *Proceedings of the 1996 International Computer Music Conference*, pages 392–395, 1996.

[Snyder2001] J. Snyder and C.L. Krumhansl. Tapping to ragtime: Cues to pulse-finding. *Music Perception*, 18:455–489, 2001.

[Temperley1999] D. Temperley and D. Sleator. Modeling meter and harmony: A preference-rule approach. *Computer Music Journal*, 23:10–27, 1999.

[Temperley2005] David Temperley. Music and probability. To be published, 2005.

[Tzanetakis2002] G. Essl G. Tzanetakis and P. Cook. Human perception and computer extraction of musical beat strength. *Proceedings of the 2002 Digital Audio Effects Conference, Hamburg*, pages 257–261, 2002.

[Vercoe1997] B. Vercoe. Computational auditory pathways to music understanding. *Perception and Cognition of Music*, pages 307–326, 1997.

[Welch2001] Greg Welch and Gary Bishop. *An Introduction to the Kalman Filter*, 2001.