Real-time Beat Tracking

Automatically Synchronising a Digital Audio Workstation to a Live Drummer

---

Presented by:
Ben James Adey

Prepared for:
A/Prof A.J Wilkinson

Department of Electrical Engineering

University of Cape Town

Submitted to the Department of Electrical Engineering at the University of Cape Town in partial fulfilment of the academic requirements for a Bachelor of Science degree in Mechatronics Engineering.

November 16, 2020

Declaration

1. I know that plagiarism is wrong. Plagiarism is to use another's work and pretend that it is one's own.
2. I have used the IEEE convention for citation and referencing. Each contribution to, and quotation in, this report from the work(s) of other people has been attributed, and has been cited and referenced.
3. This report is my own work.
4. I have not allowed, and will not allow, anyone to copy my work with the intention of passing it off as their own work or part thereof.

Signature: 
Ben Adey

Date: November 16, 2020

Acknowledgments

First of all, thank you Prof. Wilkinson for your amazing support over the course of this project. You have been so incredibly generous with your time.

I would also like my friends and family for all your help and support. Thank You Joash for your proof reading and encouragement, Nick for late night snack delivery and my wonderful family for looking after me when I didn't have time to look after myself. I am truly blessed to have all of you in my life.

I would like to thank Dylan Kulenthal for providing live drum recordings. These were of great value to the project. They show that the system can work in a live performance.

Abstract

In this project, an automatic synchronisation system BeatSync was developed. BeatSync addresses the problem of trying to synchronise a Digital Audio Workstation to a live drummer. Digital Audio Workstations are used in live performance by musicians to pay backing tracks and loops.

The problem was formulated mathematically by introducing mathematical expressions to represent the beat in music. A published automatic synchronisation system was tested, after which it was motivated to build a new automatic synchronisation system called BeatSync. The system was evaluated on a set of live drum recordings and shown to produce satisfactory results.

Contents

List of Figures

List of Tables

List of Acronyms and Terminology

DAW	Digital Audio Workstation
MIDI	Musical Instrument Digital Interface
BP	Beat Position
BBP	Bar Beat Position
IBD	Input Beat Detector
MBD	Machine Beat Detector
B-Keeper	A published automatic synchronisation system
Ableton Live	A popular Digital Audio Worksation
MainStage	The Digital Audio Workstation used in this project
Bonk	A percussive attack detector
PureData	A visual programming language

List of Symbols

θ	Beat Position
$\overline{\theta}$	Bar Beat Position
ω	Tempo
λ	Beat Division
$\overline{N}$	Number of Beats in a Bar
t	Time

Chapter 1

Introduction

1.1 Background to the study

These days, musicians don't only play with other musicians; they also play with computers. Computers have opened up a world of creative opportunities for live performers. Digital Audio Workstations (or DAWs, a kind of music production software) such as Logic and Ableton Live allow musicians to trigger electronic music sequences, loop parts of their performance, and play with arpeggiators, synthesiser instruments that repeat a pattern of notes selected by the performer. Originally invented as a tool for recording music in a studio, DAWs have evolved to become a useful musical instrument on stage. A screen shot of Ableton Live, a popular DAW, can be seen in Figure 1.1.

In addition to giving performers the tools to play with electronic music and arpeggiators, DAWs can also play backing tracks, pre-recorded musical accompaniments. An obvious example of performing to a backing track is a rap artist rapping over a hip-hop track, or a group of friends singing to their favourite song at a karaoke night. A more subtle example is a rock band playing over a pre-recorded orchestra to beef-up their sound, but without the need to tour with a full orchestra.

Before digital recording technology became commonplace, backing tracks were simply played on a tape player. DAWs give artists much more flexibility in their use of backing tracks by enabling them to repeat sections spontaneously, say by repeating a bridge for a sing-along moment. Also, sound engineers mixing the song for the audience can adjust the volume of individual elements in a backing track, for example by increasing the volume

Figure 1.1: Ableton Live, a popular Digital Audio Workstation (DAW) [2]

of the cello in an orchestral backing.

This method of using DAWs to play backing tracks is useful but a problem arises: if the band speeds up slightly, as often happens in a live performance situation, the track will begin to go out of sync with the band. Just like old tape players, DAWs are programmed to play music, whether loops, arpeggiators or backing tracks, at a fixed speed; they can't adapt to the changing tempo of the band like human musicians do. This is not a problem for a rapper or karaoke singer because there the backing track is always louder and therefore more dominant than the singer; it's easy for the performer to keep in time with the track. But a whole band playing is much louder than say, a simple piano backing track the band might want to use. As a result, the band doesn't easily notice when they begin to drift in tempo.

The solution, adopted by musicians everywhere, is called a click track: a metronome that clicks in time with the backing track or arpeggiator or loop played by the DAW. The click track is produced by the DAW and played through the band's in-ear monitors, special ear phones designed to play a customised mix of the instruments on stage so musicians can hear what they're playing (An example of in ear monitors is shown in Figure 1.1). By making the click track sufficiently loud compared to the other sounds in the performers' in-ear monitors, the band is able to stay in sync with the DAW.

In this project, we wish to investigate a different solution to the problem of synchronising to a DAW. We wish to develop a system that allows a DAW to track the beat of a live band, in particular a drummer, and automatically adjust its tempo to stay in sync. Such

Figure 1.2: A performer wearing in-ear monitors [30]

a system would enable a band to play at a naturally varying tempo, and all the while the DAW would keep in time.

1.2 Objectives of this study

The objective of this study is to develop a working automatic synchronisation system capable of synchronising a DAW to a live drummer.

1.2.1 Problems to be investigated

- Beat Tracking
- Drum Performances
- Automatic Synchronisation

1.2.2 Purpose of the study

The purpose of this study is to develop a tool that can be used by performers as an alternative to a click track to be synchronised to a DAW.

1.3 Scope and Limitations

The system will be designed to track live drums in the rock/pop genre. The types of drum patterns tested in this study are those typically found in rock/pop music. The DAW that is used in the synchronisation system is MainStage. While the system described could in theory work with other DAWs, it may require adaptation to work. The specific implemenation described in this report is solely able to synchronise MainStage.

1.4 Plan of development

Chapter 2 presents a review of previous approaches to automatic synchronisation and beat tracking. It also answers the question, "What is a beat?"

In Chapter 3, the specific design scenario being catered to in this project is described. Following that, Beat Theory is presented to understand the problem of automatic synchronisation mathematically.

Chapter 4 evaluates a working solution to the problem of automatic synchronisation and presents a new approach to the problem.

Chapter 5 describes the evaluation of the system and gives test results.

Finally, Chapter 6 draws a conclusion to the project.

Chapter 2

Literature Review

We wish to investigate the field of beat-tracking in order to develop a system to automatically synchronise a Digital Audio Workstation (DAW), a type of music creation software, to a live musician. In the following Literature Review, we will investigate what a beat is, and motivate that music has beats. We will then conduct a review of previous approaches to the problem of beat-tracking for automatic synchronisation, to learn what has already been done to solve the presented problem.

2.1 What is a beat?

Humans are repetitive. We have our tea or coffee the same way every day. We take the same route to work in the morning or the same bus home in the evening. We like to wear matching colours. We make schedules that repeat every day or week or month. Repetition is simply a part of our every day lives.

Why is it that we as humans are so interested in repetition?

In nature, we find repeating patterns all around us. In particular, we find **rhythms**, defined in the oxford dictionary as "movement marked by the regulated succession of strong and weak elements, or of opposite or different conditions" [22]. Day follows night, winter follows summer, the tide comes in then goes out. Evolutionarily, it is useful for humans to be able to notice rhythmic patterns [19]. Iverson notes that organisms are

more adaptive if they are able to "track and predict", for example by synchronising with the light and dark cycle of day and night [19].

Indeed as humans we have this innate ability of expectation. We know the end of the day is approaching, so we prepare dinner. We know winter is coming, so we gather firewood. And if we fail to accurately track and predict, we end up cold, or late for dinner.

We can see our ability to predict rhythmic patterns in the way we walk, or throw and catch balls. Many rhythms we experience day to day are instinctive or involuntary: breathing in and out, heart beating. Bispham observes that human infants are able to tell the difference between different languages based on rhythmic cues [5]. Rhythms are all around us, and we tend to notice them, whether consciously or not.

Beats exist in music

The same phenomenon of rhythm, what the Encyclopædia Britannica calls the "ordered alternation of contrasting elements" [18] is found in music. The contrasting elements are commonly called beats, and they often alternate between strong and weak [4]. Beats are moments in time, pulses repeating at regular intervals. It is the moments one might stomp their feet at in time with the music [11].

Just as people notice rhythmic patterns in nature, so too humans possess the ability to expect the rhythm, or time position of beats, in music. Our ability to predict the repeating patterns in music allows us to clap in time together, dance in choreographed performance, or simply tap our feet along to a song. We enjoy coordinated action, perhaps because, as Iverson points out, "synchronized action often leads to deep feelings of bonding and oneness between individuals" [19].

We can also observe that just as different people show varying levels of ability to catch a ball, so too different people display varying levels of ability to track and predict the beat in music. It's the reason some people are professional musicians or dancers while others are not. When we say that someone has "good rhythm", we mean that they have a strong ability to expect the time location of musical accents, or beats.

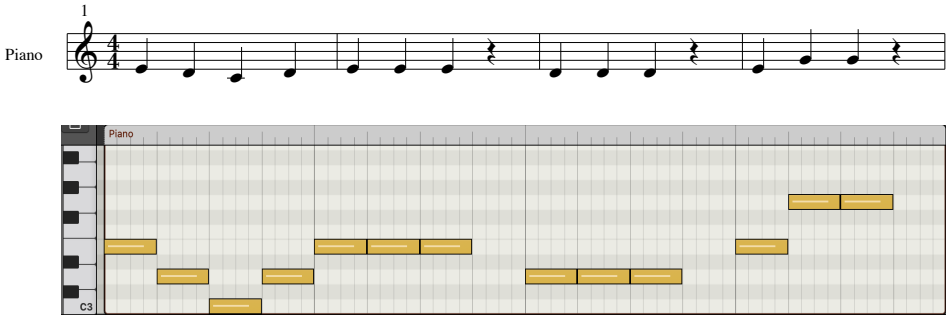


Figure 2.1: An extract of the piece “Mary Had A Little Lamb” written in music notation (top) and programmed in the DAW Logic (bottom).

Beat is perceptual

This expectation of beat is a perceptual event; it happens in the mind. According to Iverson, sound patterns only become rhythms when they interact with our brains by neural signals [19]. The same concept can easily be seen in how frequencies of light in electromagnetic radiation only become colours through interaction with our eyes and brain. Robertson agrees, stating that "the beat is a product of our perception of regularity underlying the music" [26]. Likewise, Goto and Muraoka observe that a beat doesn’t necessarily correspond to an audible event; the beat is "a perceptual concept, that a human feels in music" [14]. "Hearing the beat" is about the expectation of musical events.

Even if the beat only exists in the mind, humans are able to show agreement about where - or rather when - the beat is by clapping in time, or fist pumping the air to their favourite dance track. This same display of agreement on the time position of beats is evident in abstract representations of music, such as modern music notation. Figure 2.1 shows how the same piece of music is written in music notation and programmed in a DAW.

The *rest* symbol in music notation, 𝄽, means "don’t play anything for the duration of a beat-interval". We see in Figure 2.1 that in the DAW version of the same piece, a rest is shown simply by an absence of notes on the grid. A musician reading the music notation is able to play all the notes at the right time because they can count the rest in their mind. Likewise, the DAW has its own internal clock that counts a beat even when no note is present on the grid.

Both music notation and a DAW's representation of musical notes allows a musician or computer respectively to unambiguously reproduce a piece of music envisioned by a composer. Crucially, we see from these examples that even in the absence of a musical event, there is a moment where one expects musical events to occur. Later (in Section 3.1.1) we will develop a formal definition of "beat" around this idea of musical expectation. Beats are perceptual, and both trained musicians and clapping audiences are able to agree - perhaps with some small margin of error - that they have a definite time location. [Do I need references for all this ↑?]

2.2 Review of Previous Approaches

2.2.1 Current State of the Art

Andrew Robertson developed B-Keeper, a system that synchronises Ableton Live (a popular DAW) to a live drummer [26, 28, 27]. B-Keeper works by comparing click times from the DAW, which give the DAW's beat, to onset times from a drummer, where onsets are the moments a drum is hit. B-Keeper then performs two separate processes: phase synchronisation and tempo tracking. The phase synchronisation process tries to align the DAW's clicks with drummer onsets by adjusting the DAW's tempo by a proportion of the error between onset and click times. The tempo tracking processes tries to find the current tempo of the drummer by looking at the interval between drummer onsets. Both processes output a tempo change value which is sent to the DAW.

B-Keeper was later made commercially available as BeatSeeker, and has been successfully used by bands in live performance situations to synchronise Ableton Live. B-Keeper is the state of the art; to date, no other software has been developed to synchronise a DAW to a live performer.

While Robertson's B-Keeper is the only beat tracking system developed specifically to synchronise a DAW, research has been done in the field of beat tracking for other purposes. In the following section, various other beat tracking systems are investigated.

2.2.2 Other Approaches to Beat Tracking

The term "beat tracking" is used in literature to describe the process of identifying and/or predicting the beat times of a piece of music in an audio signal [3]. Put another way, beat trackers aim to listen to a musical excerpt and generate as output pulses at the same time locations a human might tap at in time to the music [17]. We here present a review of various approaches to the problem of beat tracking.

Tuning a non-linear oscillator

Large's proposed beat tracker works by tuning an oscillator to have period equal to the beat period of the input signal and phase such that output pulses generated by the oscillator at a particular point in its cycle are aligned in time with input onset impulses [21]. The input onset times are generated by what Large describes as a "computer-monitored piano", possibly a MIDI (Musical Instrument Digital Interface) keyboard. Adjustment only occurs at certain moments in the oscillator's cycle, meaning there is a window for change. This approach seems applicable to the goal of synchronising a DAW; if a DAW's beat could be represented as a continuous oscillator signal, it may be possible to tune it to synchronisation using Large's approach.

Autocorrelation

Davies and Plumbley determined the beat period of an input signal by autocorrelating an onset detection function and passing it through a comb filterbank matrix [10]. Likewise, Goto and Muraoka used autocorrelation to detect significant beat periods [15]. Oliveira et al [23] performed spectral flux autocorrelation on the input signal to produce a continuous periodicity function, from which peaks were selected to estimate the current beat period.

Cross Correlation

Collins' Drumtrack [9] cross correlates the input signal energy function, calculated using a Fast Fourier Transform, with impulses across an exhaustive range of possible tempos and phases. The integral of each cross correlation is taken, and the integral with the highest value is selected as the winner; the corresponding tempo and phase is chosen as

the current estimation of beat. Like Robertson's system, Drumtrack takes as input audio from a real live drummer. The system shows moderate accuracy.

Note Interval Histogram

Jensen's beat induction algorithm updates a histogram of note onset intervals every time a new onset is detected[20]. The interval with the highest frequency on the histogram is chosen to be the current estimate of beat period. Goto and Muraoka [16] Dixon [12], and Robertson [28] employ the same technique in their beat trackers.

Neural Network

Bruno and Nesi [8] built a beat tracker that used a Neural Network originally proposed by Roberts and Greenhough [25]. The Neural Network is made to behave like human tapping their foot in time to music.

Detecting Chord Changes

Most of the approaches described thus far rely on the input having strong percussive elements. Percussive events such as a drum being hit create local peaks in the audio signal, spikes that can detected by an thresholding the envelope of the signal [26]. But if the input signal is music without drums the spike created by a new note is much smaller and hard to detect purely by looking at the envelope of the signal. Choral music for examples is famously hard to beat track [10]. Goto developed a system to track music without drums by analysing harmonic information to detect chord changes [13]. Goto's system shows impressive results; 87.5% accuracy is quoted. However, unlike Collins and Robertson who processed live drummers, Goto's system was tested with audio from popular music on CDs, which has a characteristically steady tempo [14].

2.2.3 Common Features of Beat Tracking Algorithms

Onset Detection

Most of these beat tracking systems have an onset detection stage to initially extract either discrete onset times, such as in [8, 26, 12, 14] or a continuous onset detection function that has local maxima at likely onset times, such as in [10].

Detecting Kick and Snare Onsets

Robertson chose to focus on kick and snare onsets, because "strong drum events like kick and snare are the clearest indicator of where the beat falls." [26] Similary the beat trackers of Goto and Muraoka [14], and Collins [9] try to detect kick and snare onsets, relying on the fact that kick and snare are prominant features of popular music.

Weighting of Events

Several beat tracking systems weight events that are considered to be more indicative of the input beat. Robertson weights more highly drum hits at commonly expected locations, for example a kick at beats 1 and 3, or a snare at 2 and 4 (I.e. kick-snare-kick-snare, think ACDC's "Back in Black" or Michael Jackson's "Beat It"). Jensen weights older peaks less [20]. A similar weighting step is employed by Goto [13], Collins [9] and Battenberg [3].

Making Assumptions

Most of the beat tracking systems make assumptions appropriate to the expected input signal. Assumptions include:

1. **Tempo range**: The input signal is expected to be within a reasonable range. Collins for example assumes the drummer is playing between 90-190 bpm [9].
2. **Tempo is relatively constant**: This is an appropriate assumption to make since almost all pop music has a constant tempo.

3. **Time signature:** Time Signature in music theory is simply a parameter indicating how many beats are grouped in a bar. Almost all the beat trackers assume a 4/4 time signature, meaning that there are four beats in a bar (the second 4 in the time signature indicates the type of symbol used to represent a beat; e.g. 4 = ♩, 2 = 𝅗𝅥).

4. **Type of input signal**: Robertson's, Collins' and Battenberg's beat trackers are designed specifically for tracking drums [27, 9, 3]. The trackers of Goto [14] and others are designed to take as input fully mixed songs such as those on might find on a CD. While catering specifically to an isolated drum input might seem like a restricting condition, the benefit is that onset times are far more reliable. In comparison to Robertson's tracker which produces a single tempo and phase hypothesis based on onset times, the lack of onset time reliability in Goto's system necessitates multiple hypotheses t obe made in parallel and managed continuously.

5. **Genre/style**: Robertson assumes the genre is contemporary rock music [28]. By contrast, most other approaches to beat tracking are designed to work on a wide variety of genres. They might have a higher average accuracy across many genres but at the expense of robustness within a specific context.

2.2.4 Discussion

Various approaches to the problem of beat tracking have been presented. We will learn from the techniques explored, paying special attention to Robertson's B-Keeper, which has the same objective as this project. All of the techniques described rely on an understanding of music theory and we will broach this topic in the following chapter.

Chapter 3

System Identification

In this chapter the chosen design scenario is presented. Following that, the problem of automatic synchronisation is described mathematically by way of defining quantities to represent the beat. Finally, design tools will be identified to aid the development of a solution.

3.1 Design Scenario

In this section, the specific design scenario is chosen, and a DAW is selected for the system.

3.1.1 Choosing a Design Scenario

Live Band Setup

The scenario we will design for is a live band with a drummer. We assume they have access to a laptop with a DAW on it, and an external soundcard to send DAW audio output to the speakers. This equipment is typical of a band who already plays with a DAW live. An example can be seen in Figure 3.1.

Figure 3.1: A live band playing backings tracks with Ableton Live [6]. The band synchronises to Ableton Live by listening to Ableton's click track through their in-ear monitors.

Design Requirements

The automatic sychronisation system we design should use as little additional equipment as possible and be inexpensive so that it is easily accessible to both professional and amateur bands.

The system should require as little operator intervention as possible so it is easy to use.

3.1.2 Choosing an Input to Track

3.1.2.1 Choosing to Track Drums

Like Robertson [28] we choose to track the drums, for two reasons:

1. Onset detection. Drum hits have short transients making them easier to detect using signal processing techniques than say a guitar.
2. Dominant rhythm. The drums are the dominant rhythmic force in a band.
   (a) Other band members will typically lock onto the beat of the drummer.

(b) Thus, synchronisation to the drummer should allow our system to be synchronised to the whole band.

3.1.2.2 Understanding the Input: An Analysis of a Live Drum Performance

Figure 3.2 shows a collection[1] of tempo vs. time plots of a drummer playing with a live band without a click track.

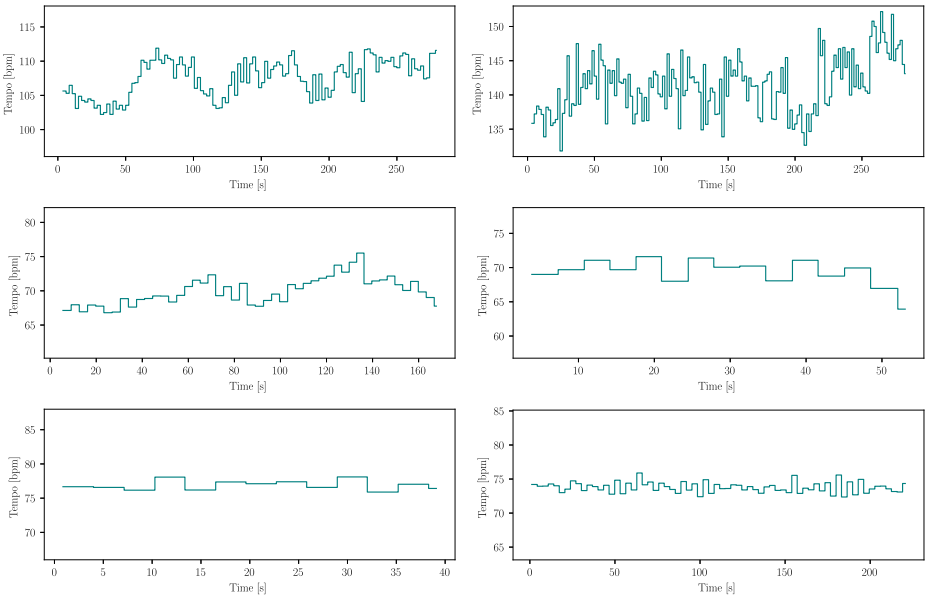


Figure 3.2: Tempo vs. time of six live drum performances.

The plots in Figure 3.2 show just how drastically a drummer's tempo can vary over the duration of a performance. A backing track playing at a fixed tempo would quickly become out of sync. It is clear from these graphs that a system capable of changing a DAW's tempo is required to synchronise the DAW.

3.1.3 Choosing a DAW

MainStage was chosen as the DAW for the automatic synchronisation system for two reasons:

1. Tempo change:

[1]The developed automatic synchronisation system is tested using these performances in the System Evaluation. They are numbered, in order from left to right then top to bottom, tracks 26-31.

MainStage's tempo can be changed by an external source using MIDI* (Musical Instrument Digital Interface) messages.

*MIDI is a protocol developed for communication between synthesisers. The protocol is covered briefly in section 3.3.

2. Affordability:

   MainStage is one of the most afforable DAWs. At the time of writing, MainStage is available for R 580[2] (full version) whereas competing DAW Ableton Live is for R 1280[3] (entry level package).

3.1.3.1 MainStage

MainStage's user interface is shown in Figure 3.3.

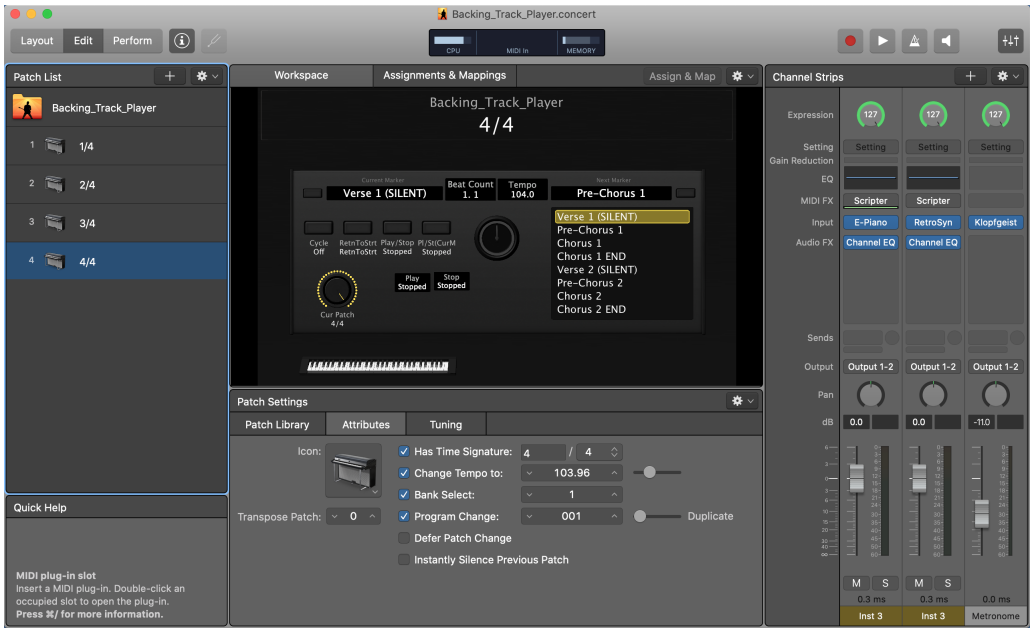


Figure 3.3: MainStage User Interface.

MainStage has many tempo-synchronous instruments and effects including:

- Arpeggiators
- Backing Tracks
- Drum machines

[2]https://www.apple.com/mainstage/
[3]https://www.ableton.com/en/shop/live/

- Tremolo
- Delay
- Low Frequency Oscillator driven filters.

Like other DAWs, MainStage provides a click track (metronome) so that musicians can keep in time with these tempo-synchronous effects.

3.2 Beat Theory

In order to develop an automatic synchronisation system, we must first understand the structure of the musical performance we wish to track. In this section we will introduce music theory notation, after which several quantities will be defined that help us understand musical structure mathematically.

3.2.1 Definitions

The terms "onset" and "beat" were earlier introduced in the Literature Review. It will be useful to have unambiguous definitions to aid our development of our automatic synchronisation system. They are formally defined as follows for the purpose of this project:

- onset: the start of a musical note or other musical event.
  - Obvious examples of an onset: the moment a drum is hit or guitar strummed or piano note played.
  - Less obvious examples of onsets: a note start present in an echo effect, the moment a string is bent up on a guitar, a low-pass filter cutoff envelope peak (often used by electronic music producers to produce a “wowow” sound from a bass synthesiser).
- the beat: the collection of discrete moments where significant onsets are expected.
  - The significance of an onset is determined by its relative position in a bar of music, a concept covered in this next section.

- A single *beat* is simply a single discrete moment where significant onsets are expected.

The presence of onsets in an audio signal (acoustic or electric) creates the perception of the beat. And the perception of beat leads to an expectation that onsets will occur in the future. It follows that if we say we "hear the beat", it means we expect onsets at discrete moments in the future.

3.2.2 Introduction to Music Notation

Modern music notation is used by musicians all over the world. It is a way to abstract a musical performance so it can be shared with others. The staff in music notation is the blank template onto which notes are added. Figure 3.4 shows both an empty staff and one with a melody written on it.

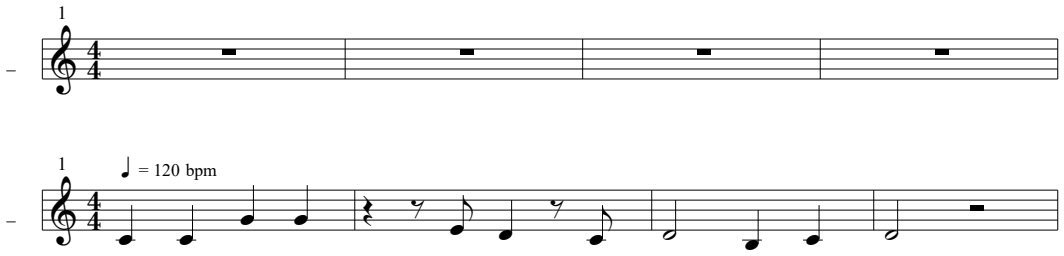


Figure 3.4: An empty staff (top) and a staff with an example melody on it (bottom).

The essential elements of a staff are explained explained below:

Time signature 

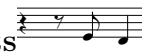
- The lower number indicates which note value is used to dictate a note with inter-beat-interval (IBI) duration.
  - In this case, the 4 indicates that a *quarter note*, symbol ♩, has IBI duration.
- The upper number indicates the number of beats in a bar, and is typically 4 for the vast majority of popular music.

- Bar boundaries (the first beat of a bar), are where new sections of a song begin such as the start of a verse or chorus. They are also the moment chord changes will typically happen.
- Each bar has precicesly equal duration.
- Patterns will repeat in integer multiples of a bar. Repeating patterns are commonly called "riffs", but may also be called a "loop" or "groove".
- When musicians count the beats in time to music, they count from the first beat of the bar. For example: "**1** - 2 - 3 - 4 - **1** - 2 - 3 - 4 - **1** - 2 - ...". A click track indicates the first beat of the bar by ticking at a higher frequency.

Tempo ♩ = 120 bpm

The tempo gives the IBI, and is expressed in beats per minute (bpm). In this example, a tempo of 120 bpm means that a quarter note is played every 0.5 seconds.

- In a live performance, musicians will often indicate the tempo by counting "1-2-3-4-" after which the whole band comes in on the "1" of the next bar. Alternatively, as is common in rock bands, the drummer will "count-in" the band by clicking drum sticks four times.

Notes and Rests 

A composer writes *notes* using different symbols to indicate different note lengths. For example, a quarter note is given by the symbol ♩, and has double the duration of an eigth note, ♪.

A composer writes a *rest* as a placeholder, indicating a moment when the musician should be silent. A quarter note rest is 𝄽 and an eighth note rest is 𝄾.

A full table table of note and rest values is included in Appendix B.1.1.

These conventions, such as using the quarter note symbol or grouping four beats in a bar, are not hard and fast rules that a composer must follow. They have developed to be the standard simply as a result of popular use, and are indicative of the fact that songs in popular music have a similar structure. An example of an exception to grouping four beats in a bar is waltz music, which has three beats in a bar (count "**1** - 2 - 3 - **1** - 2 - 3 - **1** - 2 - ...").

3.2.3 Mathematical Expresssions to Represent Beat

Music notation is a useful way for humans to read and understand the structure of music. However it is symbolic, and is unsuitable for computers. We will require a computer program to run our automatic synchronisation system, and thus need to define the structure of music using math expressions. These can be later be used to write code in a programming language such as Python. Here we present definitions and conventions of several quantites that together can express the nature of the expected beat in a musical performance. We will use the short piece of music in Figure 3.4 as a case study.

3.2.3.1 Beat Position θ

Beat Position (BP) is a measure of how much a musical performance has progressed.

- BP is a scalar quantity, a real number.
- BP has symbol θ and is measured in beats.
- A *beat* occurs at every integer value of BP.
  - The first beat of a song is at $\theta = 1$ beats ; the second beat of a song is at $\theta = 2$ beats and so on.

Note onsets can then be said to occur at a particular BP. In our example piece, the first note onset occurs at $\theta = 1$, and the second note onset at $\theta = 2$. This can be seen in Figure 3.5.

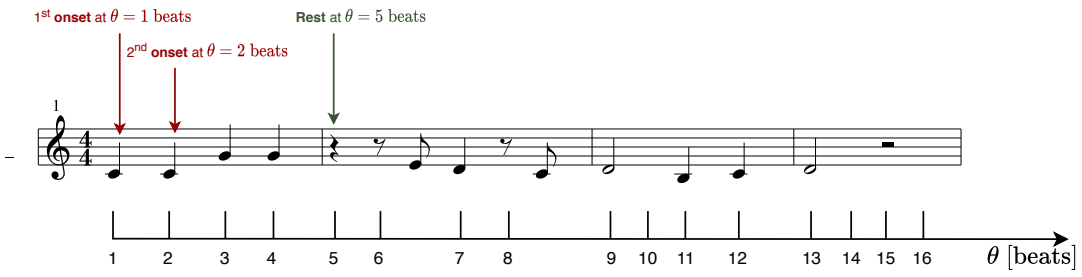


Figure 3.5: Note onsets mapped to Beat Position.

As stated earlier, beats, which occur at integer values of BP, are characterised by being positions where we expect note onsets, thought note onsets and beats don't always coincide. At $\theta = 5$ for example, there is no note onset, but instead a rest (see Figure 3.5).

How do we relate Beat Position to time?

If the musical performance has a constant tempo, we can say that BP is linearly related to time. We define the **constant tempo BP function**:

$$\theta(t) = \omega \cdot t + c \tag{3.1}$$

where

- ω is the tempo in beats per second, or bps. (e.g. 120 bpm = 2 bps).
- t is absolute time in seconds.
- c is a BP-shift constant calculated so that the BP function intercepts the point where $\theta = 1$ and $t = t_0$
  - t_0 is the absolute time of the first beat of the performance.
  - Then $c = 1 - \omega \cdot t_0$

Example

Let's assume a performance of the piece given in Figure 3.4 begins at $t = 0$ s, and the tempo is constantly 120 bpm as prescribed by the composer. Then the BP function is determined as follows:

$$\begin{array}{l} \omega = 2 \text{ bps} \\ c = 1 - \omega \cdot t_0 = 1 \text{ beats} \end{array} \quad \rightarrow \quad \theta(t) = 2t + 1$$

The BP function is plotted in Figure 3.6.

As seen in Figure 3.6 we can plot the note onsets from the example musical piece on top of the BP function. Each note onset has a time location and a BP. We can now make observations such as: the fifth note onset occurs half way between two beats. We can also calculate when a certain BP will occur by inverting equation 3.1:

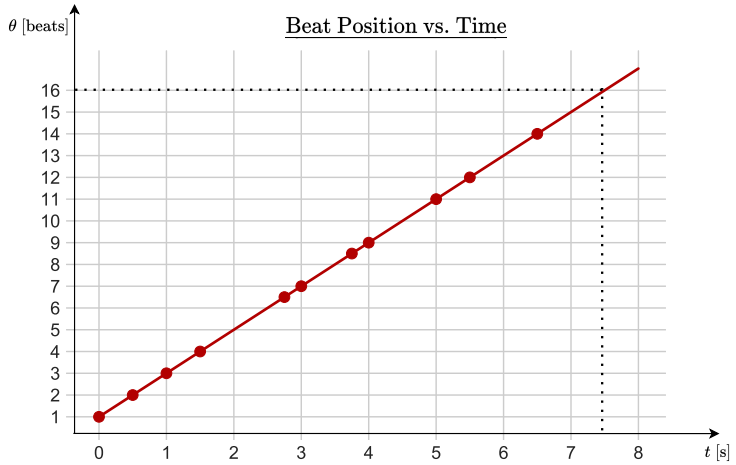


Figure 3.6: Beat Position function of the example piece in Figure 3.4.

$$t = \frac{1}{\omega}(\theta - c) \tag{3.2}$$

We can calculate for example that the 16^{th} beat of the piece will occur at $t = \frac{1}{2}(16-1) = 7.5$ s. We can observe this visually by the dotted line in Figure 3.6.

3.2.3.2 Bar Beat Position $\overline{\theta}$

Bar Beat Position (BBP) is the BP of a musical performance counted from the first beat of a bar. It has symbol $\overline{\theta}$ and is measured in beats. BBP is a function of BP:

$$\overline{\theta}(\theta) = [(\theta - \theta^1) \ \% \ \overline{N}] + 1 \tag{3.3}$$

where

- θ^1 is a BP where it is known that the BBP is one.
  - In our example, the first beat of the piece is at BBP $= 1$, therefore $\theta^1 = 1$. In some cases a song will begin with what is called an *anacrusis*, whereby the first bar is shortened and the first beat of the song is at a BBP greater than 1.
- % is the modulo operation: $a \ \% \ b$ gives the remainder of $\frac{a}{b}$.

- $\overline{N}$ is the number of beats in a bar, given by the upper number in the time signature. and in our example is 4.

Note onsets can be mapped to BBP just as with Beat Position (see Figure 3.7).

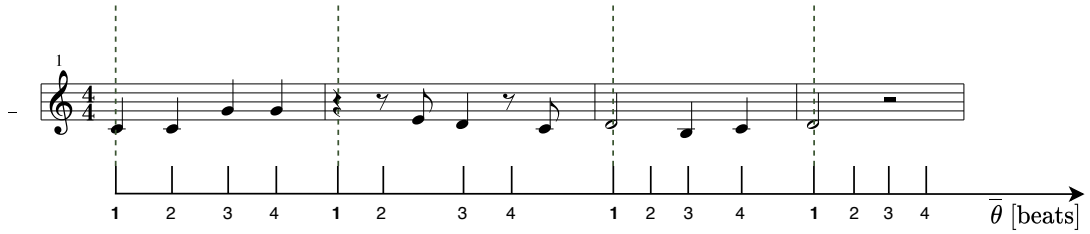


Figure 3.7: Note onsets mapped to Bar Beat Position. Bar boundaries are indicated by dotted lines.

The BBP function for the example piece is shown in Figure 3.8.

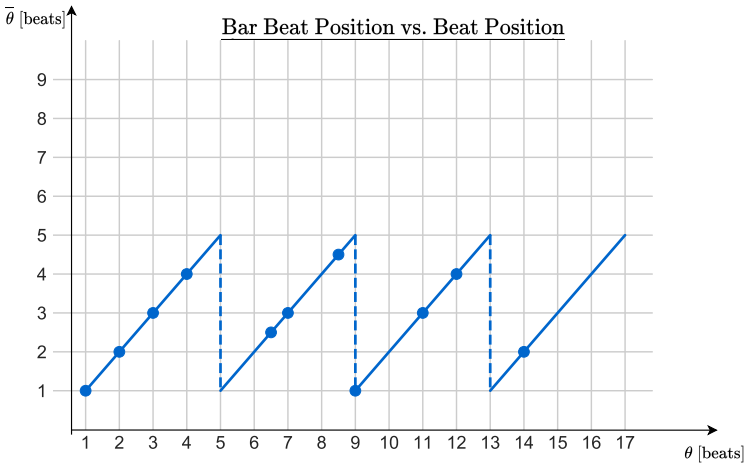


Figure 3.8: Bar Beat Position function of the example piece in Figure 3.4. Bar boundaries are indicated by dotted lines.

We can see in Figure 3.8 that BBP progresses in the same way that a musician counts: "**1** - 2 - 3 - 4 - **1** - 2 - 3 - 4 - **1** - 2 - ...". This is a useful quantity because it is common for musical events to be synchronised to the start of a bar. In MainStage, when a performer triggers a backing track, the track does not begin playing immediately. Instead, MainStage begins playing the backing track at the start of the next bar, synchronised to the first tick of MainStage's click track.

3.2.3.3 Beat Division λ

Beat division is the number of onsets expected in the same amount of time as the IBI. It can be calculated by finding the ratio of the IBI note value to the shortest note value in a piece.

1. IBI note value is $\frac{1}{[\text{Time Signature Lower Number}]}$. In our example, IBI note value is $\frac{1}{4}$ (quarter note).
2. Shortest note value is simply the minimum note value found in a piece. In our example shortest note value is $\frac{1}{8}$ (eighth note).

Thus, for the example piece $\lambda = \frac{1/4}{1/8} = 2$. In other words, we expect at maximum two note onsets in the same amount of time as the IBI. Commonly, music pieces have a beat division of 2, 3 or 4.

3.2.3.4 Sub Beat Position ϕ

Sub Beat Position (SBP) is a measure of how much a musical performance has progressed, measured in sub beats. SBP is a function of BP:

$$\phi(\theta) = \lambda \cdot \theta - 1 \tag{3.4}$$

We can then say that a *sub beat* occurs at every integer value of SBP. Using equations 3.1 and 3.4, we can define exhaustively the entire collection of discrete moments (time locations) where onsets are expected:

$$t[\phi] = \frac{1}{\lambda \cdot \omega}(\phi - \lambda \cdot c)\ , \quad \phi = 1, 2, ..., \tag{3.5}$$

Equation 3.5, representing the collection of all sub beats, is plotted for the range $\phi = [1; 31]$ on top of the SBP function for the example piece in Figure 3.9.

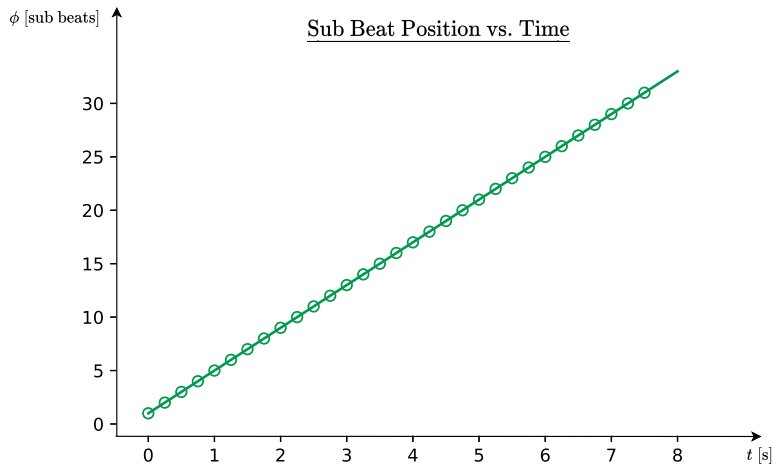


Figure 3.9: Sub Beat Position function indicating all time locations where onsets are expected for the example musical piece given in Figure 3.4.

3.2.3.5 Applying Beat Theory Expressions

In this section, we have described the problem of automatic synchronisation mathemeti-cally by way of introducing mathematical expression to represent the beat of an instance of music. Both MainStage and the drummer have their own Beat Position θ, and associated tempo ω, Bar Beat Position $\overline{\theta}$ and Beat Division λ.

For this project, MainStage's collection of beat expressions will be represented by the subscript m for *machine*: θ_m , ω_m , $\overline{\theta}_m$, λ_m.

The drummer's beat will be represented by the subscript i for *input*: θ_i , ω_i , $\overline{\theta}_i$, λ_i.

3.2.4 The Problem

Earlier MainStage's instruments were described as being "tempo-synchronous." It is indeed true that they are tempo synchronous, but they are also **beat-synchronous**. When you press a key on an arpeggiator in MainStage the arpeggiator only begins playing a note at the next beat[4], and then it continues to play notes on each subsequent beat until you release the key. When you use a Low Frequency Oscillator driven filter (think "wowowowow"), the oscillator's peaks occur on MainStage's beats.

MainStage's beats occur when its Beat Position θ_m is at an integer value, and MainStage's

[4]Or sub beat. In MainStage, you can set the beat division of an arpeggiator.

Beat Position θ_m increases at a linear rate determined by its tempo ω_m. MainStage keeps track of its Beat Position internally. As described earlier, a perfomer can hear MainStage's beat by listening to the click track MainStage produces. Each click occurs at one of MainStage's beats,

In order to synchronise MainStage to the input, MainStage's tempo must be adjusted so that it's Beat Position is aligned with that of the input. But first, the Beat Position of both MainStage and the input must be determined.

In summary, we require:

- A means to determine the Beat Position of the input θ_i
- A means to determine the Beat Position of MainStage θ_m.
- Some way of deciding how to change MainStage's tempo so that its Beat Position coincides with the input.

3.2.4.1 A Car Analogy

The problem of automatic synchronisation can perhaps best be understood by way of an analogy:

Two cars, Car A and Car B, are placed side by side on a straight road of infinite length (see Figure 3.10).

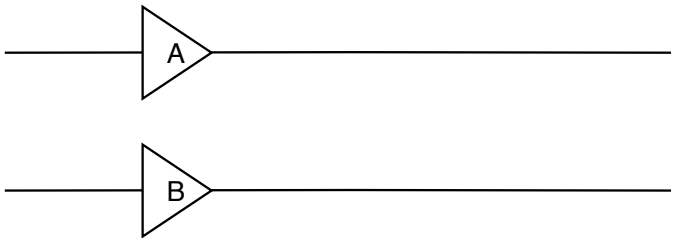


Figure 3.10: Car analogy: Car A and Car B side by side.

Car A begins moving at a constant speed v_a so that its position x_a begins increases linearly (see Figure 3.11).

We want Car B to be side by side with Car A (i.e. at the same position). We can make Car B catch up to Car A by setting its speed v_b (see Figure 3.12).

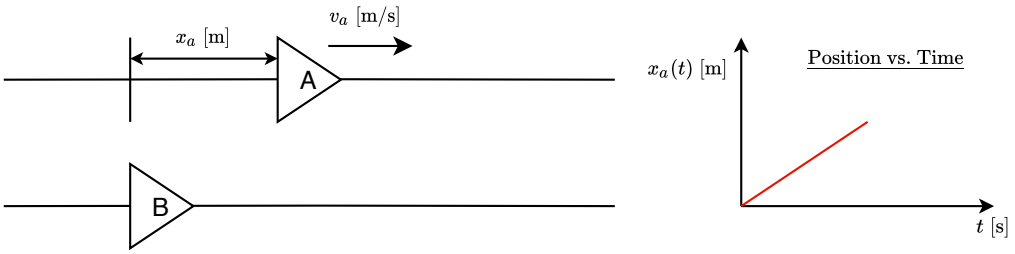


Figure 3.11: Car analogy: Car A's position increases linearly.

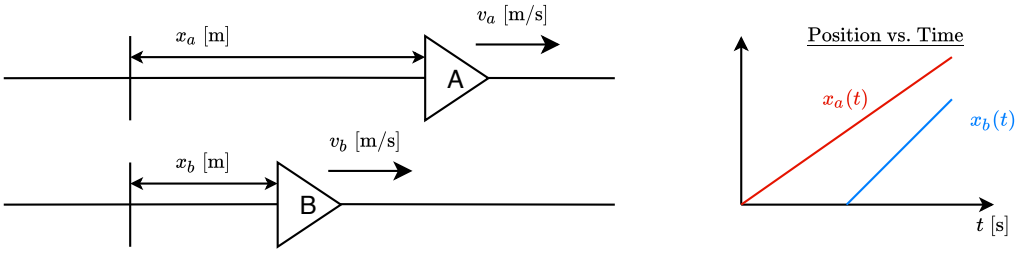


Figure 3.12: Car analogy: Car B catching up to Car A to be side by side.

In our design scenario, the input (drummer) is analogous to Car A, increasing in Beat Position at some tempo. MainStage is analologous to Car B. We want MainStage to be "side by side" with the input. It is not enough for MainStage to have the same tempo as the input. The problem is visualised in Figure 3.13.

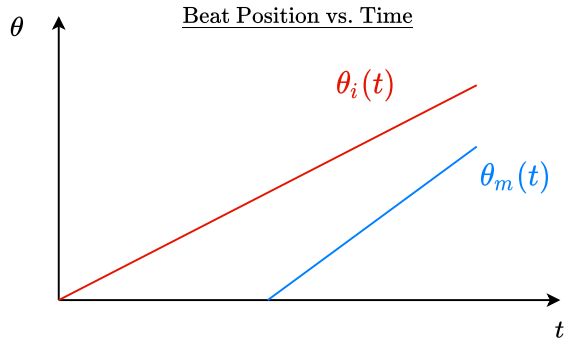


Figure 3.13: Machine and Input Beat Position functions.

3.3 Design Tools

In this section, the design tools used for development of the automatic synchronisation system are briefly introduced. The software version are also given.

3.3.1 Python

Python was chosen as the programming language because it allowed rapid development of a software solution. Python version 3.7.6 was used.

3.3.2 PureData

PureData is a visual programming language that makes it easy to interact with digital audio streams. In PureData, complex systems can be created rapidly using a drag-and-drop style interface. Pure Data will be used in a section of the automatic synchronisation system.

3.3.3 MIDI

MIDI (Musical Instrument Digital Interface) was originally developed as a protocol to enable hardware synthesisers to interact with one another. Today, MIDI is also a software protocol. MIDI is the only way to communicate with MainStage using software (i.e. without clicking on the screen) is via MIDI (Musical Instrument Digital Interface). It is also ideal since it is easy to implement and has low latency. In this project, MIDI will be used in the following ways:

- MIDI Ports:
  - Messages will be sent to MainStage and recieved from MainStage via a MIDI port, a software connection over which MIDI messages are sent.
  - MIDI ports will also be used to communicate between different sections of the automatic synchronisation system.
- MIDI Files:
  - MIDI messages can be stored in MIDI file from which they can be played back sequentially.
  - In this project, MIDI files are used to store recordings of drum patterns and input data.

3.3.4 Computer

The system was tested on a MacBook Pro (Retina, 13-inch, Early 2015) with the following specifications:

Processor 2.7 GHz Dual-Core Intel Core i5

Memory 8 GB 1867 MHz DDR3

Graphics Intel Iris Graphics 6100 1536 MB

3.3.5 MainStage

MainStage version 3.4.4 was used.

Chapter 4

System Design

4.1 Evaluating an Existing Algorithm: B-Keeper

A good place to start is to investigate an already developed system that has been used in practice. Early in the project, the B-Keeper algorithm was tested.

B-Keeper, designed by Andrew Robertson, is a working automatic sychronisation system made to control the DAW Ableton Live. It was not possible to test the actual B-Keeper due to a lack of access to Ableton Live. However, Robertson's thesis paper [26]describes the algorithm in detail, and so it was possible to write a program that copied B-Keeper for use with MainStage.

The B-Keeper algorithm is described briefly in this section. A brief, informal evaluation is also included after which a motivation is given for the development of a new system.

4.1.1 Algorithm Description

The B-Keeper algorithm works by comparing click times from the DAW to onset times from a drummer. For each onset received from a drummer, two procedures are performed:

1. Tempo tracking:

   The tempo tracking process tries to identify the most likely tempo at the current

onset by looking at the interval between the current onsets and previous onsets. The Bar Beat Position of the current onset is estimated by adopting the Bar Beat Position of the nearest DAW click. The Bar Beat Position of the nearest DAW click is assumed by counting up from the first click received by the DAW.

2. Synchronisation:

   The synchronisation process tries to identify how much the DAW tempo needs to change in order to become what Robertson describes as being "in phase" with the drummer.

   (a) It evaluates incoming onsets based on their accuracy, and only considers onsets that are accurate enough.

Both processes output a tempo change value that is sent to the DAW.

4.1.2 Python Implementation

The algorithm was implemented in a Python program. The algorithms and time keeping procedure used to develop the Python program are included in flowchart form in Appendix B.2.1

4.1.3 Qualitative Evaluation

The B-Keeper algorithm works. It was tested with some basic drum patterns and showed a good ability to track the tempo fluctuations of a drummer. There is however one aspect of the B-Keeper algorithm that makes it less suitable for MainStage: too many tempo changes. The B-Keeper algorithm makes a tempo change at the moment an onset arrives (as quickly as possible). It cannot make the tempo change later. The problem is that when there are too many onsets, and B-Keeper makes a tempo change for each of those onsets, MainStage exhibits undesirable behaviour. In particular, there are audio drops and "glitches" if MainStage is playing a backing track. The audio glitches are due to the fact that MainStage has to re-calculate its audio output sample stream (re-buffer) every time it receives a tempo change.

It was therefore deemed necessary to devise a system that could make tempo changes at a time other than at an onset. Such a system would be able to receive many onsets

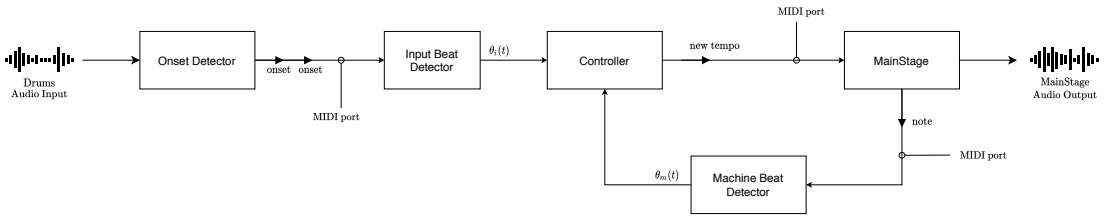


Figure 4.1: BeatSync whole system diagram.

but make only a single tempo change in the same amount of time that the onsets were received.

In the following section, the automatic synchronisation system devised for this project is presented. The system is called BeatSync.

4.2 BeatSync: A New Approach to Automatic Synchronisation

In this section, a new approach to automatic synchronisation of a DAW is presented. The approach is based on an understanding of Beat Theory, the collection of mathematical expressions defined earlier in section 3.2.

4.2.1 System Overview

Earlier the problem of automatic synchronisation was summarised by saying that we need a means to determine the input Beat Position θ_i, a means to determine the machine Beat Position θ_m and a way of deciding how to change machine tempo ω_m. These tasks will be achieved respectively by an Input Beat Detector, Machine Beat Detector, and Controller. In addition, based on Robertson's B-Keeper, an Onset Detector will be used to detect the times that a drum is hit. The whole system diagram is shown in Figure 4.1.

The Input Beat Detector, Machine Beat Detector and Controller are implemented in a Python program. The Onset Detector is implemented in a PureData program. In the following subsections, the design of each subsystem is described.

4.2.2 Onset Detector

As we have learnt from Beat Theory in Section 3.2.3.1, we can use onset times to identify the input BP function (input beat position and tempo). Like Goto and Muraoka [14], Robertson [28] and Collins [9], we will focus on detecting kick and snare drum onsets because, as Robertson notes, "strong drum events like kick and snare are the clearest indicator of where the beat falls." Put another way, the kick and snare pattern serves as a strong foundation on which most drum patterns are built [7].

We therefore desire an onset detector that can reliably produce an output message every time the kick or snare drum is hit and it must distinguish between hits from the two drums.

4.2.2.1 Choosing Microphones

The first step in onset detection is to capture the sound produced by the drums and convert it to a digital signal. Since we want a solution that uses as little extra equipment as possible, it would be ideal to capture audio using the laptop's built-in microphone. There are however two major problems with this approach:

1. **Background noise**. The laptop's microphone picks up the kick and snare well, but it also picks up other instruments and drums we are not interested in. In a live performance setting, there are all kinds of other loud sounds on stage apart from the drums such as guitar and bass amplifiers. We need a microphone that can pick up the kick and snare but adequately reject background noise from other sound sources.
2. **Hard to distinguish between kick and snare**. The laptop's microphone outputs a single channel of audio, so would have to cover both the kick and the snare. It is difficult to reliably tell the difference between a kick or snare hit simply by looking at a single waveform. Two seperate audio channels are required.

We thus require two seperate microphones capable of rejecting background noise. We chose to use contact microphones because, unlike regular air microphones, contact microphones produce a signal from mechanical vibration. They are almost completely insensitive to vibrations in the air [1]. Contact microphones are not typically used to record

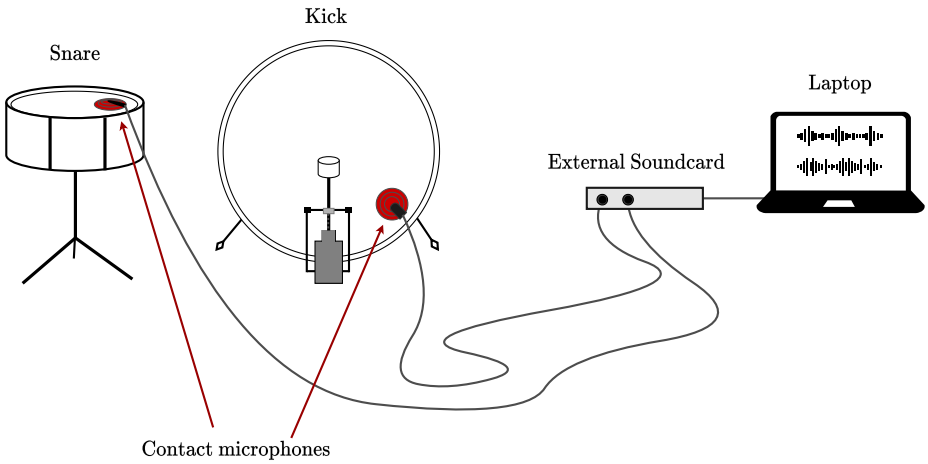


Figure 4.2: Contact microphones attached to kick and snare drum.

drums simply because the sound produced from a contact mic is not that pleasant compared to the sound captured by traditional dynamic microphones. We are not interested in the musical quality of the signal however, so contact mics are appropriate for our application. Contact mics are also much cheaper than the dynamic microphones typically used for recording drums and so fit our design requirement that the sytem be inexpensive.

Note that while the set-up described here uses contact microphones, in reality the onset detector will also work with dynamic microphone inputs, provided they are placed close enough to the kick and snare to be sufficiently isolated from other sounds. The system will thus be suitable for bands that already use dynamic mics.

The contact mics were attached to the kick and snare drum and connected to a two-channel Analogue to Digital Converter (ADC) on an external soundcard as shown in Figure 4.2.

4.2.2.2 PureData Program

The digital audio signal from the microphones is processed by a working onset detector, `Bonk`. `Bonk` is a percussive onset detector designed by Miller Puckette for the visual programming language, PureData [24]. It works by calculating the growth in signal power across 11 frequency bands. If the growth exceeds a given threshold, `Bonk` outputs a 'bang', a simple message that triggers other processes in PureData. The same onset detector was used by Robertson [27]. The complete program in PureData is shown in Figure 4.3.

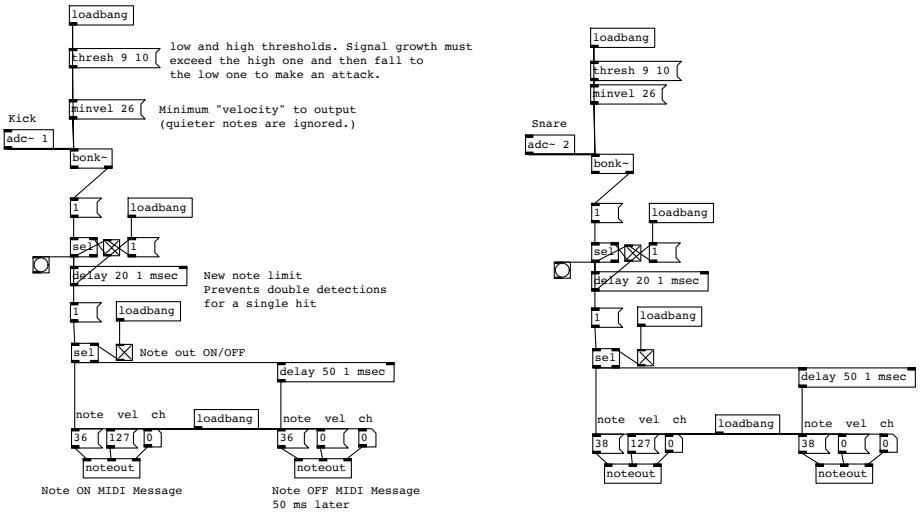


Figure 4.3: Onset Detector program in PureData visual programming language.

The PureData program has the following features:

- ADC digital input from soundcard.
- Onset Detection using Bonk
- Threshold values which determine how sensitive the Onset Detector is to input sounds.
- Debounce limit to prevent double detection for a single onset.
- MIDI `note on` message output:
  - Kick and snare onsets output MIDI note numbers 36 and 38 respectively. These note numbers were chosen because they are the standard kick and snare numbers used by DAWs. The Onset Detector output can be connected to a DAW and the DAW will play kick and snare samples[1].
- MIDI `note off` message output:
  - These are included so that there are no notes left "hanging" in the case that the Onset Detector is connected to a DAW for audible feedback.
  - The `note off` messages are not required by the BeatSync system.

[1]This may require a virtual instrument to be installed. Virtual drum instruments come pre-installed with MainStage.

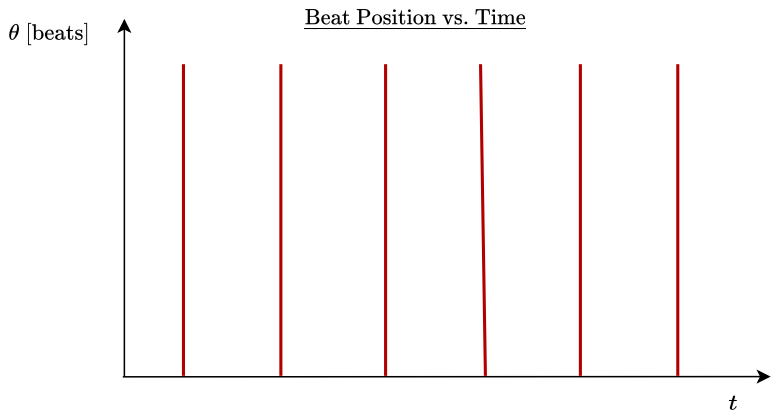


Figure 4.4: Incoming onsets represented as impulses on a plot of Beat Position vs. time.

The code (set of blocks) on the left in Figure 4.3 is identical to the code on the right except for the MIDI note numbers they output. They process kick and snare respectively.

4.2.2.3 Verification

The Onset Detector subsytem was verified by connecting its MIDI output to Logic. Every time a drum was hit, Logic played a drum sample, kick or snare depending on the note number. It was noticed that there was a slight delay between hitting the drum and an onset being detected. It is necessary to correct for this latency in later subsystems.

The audible feedback from Logic was helpful in designing the Onset Detector. By playing the drum kit with varying amounts of force and hearing the drum samples from Logic it was possible to tune the Onset Detector's threshold parameters to be ideal for the drum input from the contact microphones. The threshold values were set so that the Onset Detector is sufficiently responsive to drum hits but doesn't produce any false onsets (onsets in the event of no drum hit).

4.2.3 Input Beat Detector (IBD)

The Onset Detector provides a means to detect onset times, but the Beat Position of those onsets is still unknown. If we represented the incoming onsets on a plot of Beat Position vs. time, they would appear as impulses. (see Figure 4.4).

In the example given in Section 3.2.3.1, it was possible to produce a BP function from

onsets because the BP of each onset was known from the staff. In our design case, the staff notation of a perfomance will not be available. It will however be assumed that the user is able to provide the following information about a performance:

1. Number of beats per bar $\overline{N}_i$
2. Beat division λ_i
3. Initial tempo ω_{init}

The first two can be typed into the BeatSync program. The initial tempo is given by the drummer *counting-in* BeatSync by tapping a drum stick on the kick or snare four times (or three times if $\overline{N}_i = 3$ etc.). This information is easy for a band to provide and would be known from rehearsal.

The rest of this section contains a description of the Input Beat Detector (IBD) algorithm. Following that a weighting system is introduced.

4.2.3.1 Algorithm

Onsets arrive at the IBD already time stamped. For the n^{th} input onset at time $t = t_{ons}[n]$, two steps are performed:

1. Beat Position $\theta_{ons}[n]$ is determined.
2. An estimate of the input BP function ($\theta_i(t) = \omega_i \cdot t + c_i$) is produced by estimating:
   (a) Input tempo ω_i [in beats per second]
   (b) Input BP-shift constant c_i[in beats].

   Note that a constant tempo BP function is assumed for simplicity's sake.

First Onset

1. Determine Beat Postion:
   (a) First onset is assumed to be at BP of 1: $\theta_{ons}[1] = 1$ (first hit of the count-in).

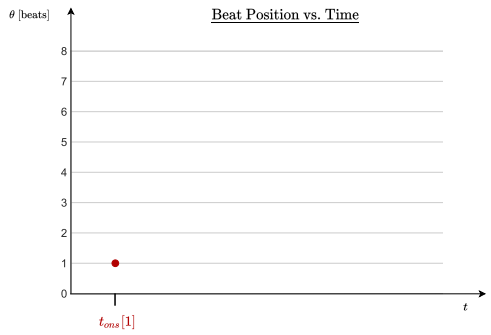


Figure 4.5: Input Beat Detector receiving first onset.

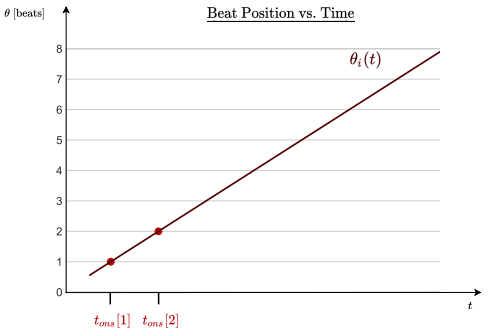


Figure 4.6: Input Beat Detector receiving second onset.

2. Estimate the input BP function (see Figure 4.5):
   (a) No tempo or BP-shift constant estimate yet.

Second Onset

1. Determine Beat Position:
   (a) Second Onset is assumed to be at BP of 2: $\theta_{ons}[2] = 2$ (second hit of the count-in).
2. Estimate the input BP function:
   (a) BP function is simply a straight line through the two recorded onset points:
      i. $\omega_i = \frac{\theta_{ons}[2] - \theta_{ons}[1]}{t_{ons}[2] - t_{ons}[1]}$
      ii. $c_i = \theta_{ons}[1] - \omega_i \cdot t_{ons}[1]$

Third Onset or Later

1. Determine Beat Position by *quantisation* (find the closest sub beat):

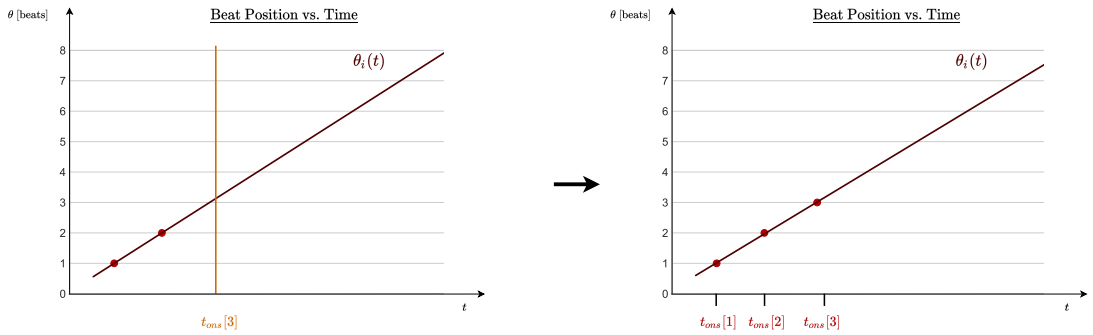


Figure 4.7: Beat Position of third and subsequent onsets is determined by quantisation to the closest sub beat.

(a) First, calculate the esimated BP $\tilde{\theta}_{ons}$ at time t_{ons} given the Input Beat Detector's current estimate of the input BP function:

i. Use equation 3.1: $\tilde{\theta}_{ons} = \theta_i(t_{ons}) = \omega_i \cdot t_{ons} + c_i$

(b) Then, find the BP of the nearest sub beat:

i. $\theta_{ons} = \text{round}\{(\tilde{\theta}_{ons} - \text{int}\{\tilde{\theta}_{ons}\}) \cdot \lambda_i\} \cdot \frac{1}{\lambda_i} + \text{int}\{\tilde{\theta}_{ons}\}$

where:

- $\text{int}\{x\}$ gives the integer part of x (e.g. $\text{int}\{3.72\} = 3$)
- $\text{round}\{x\}$ gives x rounded to the nearest integer (e.g. $\text{round}\{3.72\} = 4$)

Figure 4.7 shows the process of quantisation.

2. Estimate the input BP function by finding the *line of best fit* through the recorded onsets:

(a) The standard Linear Least Squares Regression[2] formulas will be used, adapted slightly to include a weighting parameter for each onset. In addition, an evaluation window is set: only onsets within the last θ_{window} beats will be considered in the calculation. The choice of weighting parameters and window width θ_{window} is described in section 4.2.3.2. The BP function is calculated as follows:

$$\omega_i = \frac{\widehat{n}\sum W\widehat{t}_{ons}\widehat{\theta}_{ons} - (\sum W\widehat{t}_{ons})(\sum W\widehat{\theta}_{ons})}{\widehat{n}\sum W(\widehat{t}_{ons})^2 - (\sum W\widehat{t}_{ons})^2} \tag{4.1}$$

$$c_i = \frac{\sum W\widehat{\theta}_{ons} - \omega_i(\sum W\widehat{t}_{ons})}{\widehat{n}} \tag{4.2}$$

[2]https://www.mathsisfun.com/data/least-squares-regression.html

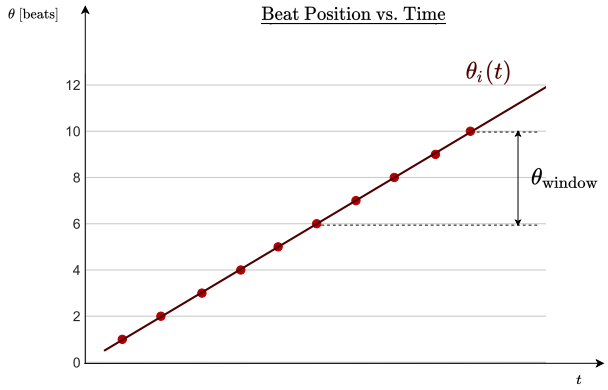


Figure 4.8: Input Beat Detector line of best fit calculation with BP evaluation window θ_{window}.

where:

- $\widehat{t}_{ons}$ and $\widehat{\theta}_{ons}$ are the respective times and BP values of the set of onsets in the evaluation window.
- $\widehat{n}$ is the number of onsets in the evaluation window.
- W is a set of weighting parameters, one for each onset in the evaluation window.

The line of best fit with the Beat Position window of evaluation is shown in Figure .

4.2.3.2 Weighting Parameters and Window Width

The choice of weighting parameters is based on Robertson's weighting system in the B-Keeper algorithm. The weighting system follows three basic rules:

1. Onsets that are more accurate are weighted higher
   (a) Favouring accurate onsets ensures stability of tempo.
2. Kick onsets are weighted higher than snare events.
   (a) Kick drum patterns are more likely to be consistent and definite. By contrast, snare patterns often produce a continuous flurry of onsets as a result of the rolling motion of a drum stick on the snare.
3. Onsets that are on the beat are weighted higher than onsets off the beat.

(a) Onsets at $\theta_i = 1, 2, 3, 4$ are typically more indicative of the beat than onsets at say $\theta_i = 1.5$ or $\theta_i = 2.333$.

Robertson's B-Keeper algorithm chooses to ignore onsets with a low weighting. In contrast, the Input Beat Detector includes all onsets in the line of best calculation, but lower weighted onsets have less effect on the predicted BP function. The benefit of including all onsets in the line of best fit calculation is that the Input Beat Detector is responsive to low weighted events in the absence of higher weighted events. For example, if only the snare has been played for a while, then the line of best fit will be based purely on those snare onsets. If onsets are only played off the beat by choice of the drummer, the Input Beat Detector will become responsive to the changing tempo of those onsets.

4.2.4 Machine Beat Detector (MBD)

The Machine Beat Detector (MBD) has the task of estimating the machine BP function ($\theta_m(t) = \omega_m \cdot t + c_m$) as accurately as possible. The process is similar to the IBD.

4.2.4.1 MainStage Setup

A script was written in MainStage to play a sequence of notes synchronised to MainStage's beat, and send them over a MIDI port to BeatSync. The notes are played at a known beat division $\lambda_m = 2$. The script, and steps to set up MainStage for use with BeatSync are included in Appendix A.1.1. It is assumed that the number of beats per bar in MainStage has been set by the user to match the input $\rightarrow \overline{N}_m = \overline{N}_i$. The note number of each MIDI message from MainStage identifies its Bar Beat Position. The notes are not audible in MainStage.

4.2.4.2 Tempo is Sometimes Unknown

At most times, machine tempo ω_m is known and definite because BeatSync has already sent a tempo change to MainStage. When the tempo is known, the MBD only needs to calculate c_m. An example where this is not the case is if the user decides to press play on MainStage before they start drumming. A band might decide to do this because they

want the backing track to start first. In order to be as flexible as possible, the MBD is designed to work for both cases.

4.2.4.3 Algorithm

Onsets arrive at the MBD already timestamped, and with a known Bar Beat Position (BBP).

For the n^{th} machine onset with time and BBP $(t_{ons}[n], \overline{\theta}_{ons}[n])$, the following steps are performed:

If First Onset:

1. Determine Beat Position:
   (a) The BP of the first onset is equal to its BBP $\rightarrow \theta_{ons}[1] = \overline{\theta}_{ons}[n]$
2. Estimate the machine BP function:
   (a) If tempo is known, $\omega_m =$ [known tempo] and $c_m = \theta_{ons}[1] - \omega_m \cdot t_{ons}[n]$, otherwise no estimate.

Second Onset or Later:

1. Determine Beat Position:
   (a) Beat Position increments at a known beat division: $\rightarrow \theta_{ons}[n] = \theta_{ons}[n-1] + \frac{1}{\lambda_m}$
2. Estimate the machine BP function:
   (a) If tempo is unknown, use equations 4.1 and 4.2 (excluding the weighting parameters) to calculate ω_m and c_m.
   (b) If tempo is known, only calculate the BP-shift constant c_m using equation 4.2.

The reason for using the line of best fit equations is to reduce error due to some small amount of timing jitter. A fixed evaluation window width $\theta_{m-\text{window}} = 6$ [beats] is chosen by trail and error.

4.2.4.4 Tempo Changes

Whenever BeatSync sends a tempo change to MainStage, it also send the new tempo to the MBD. The MBD then knows the definite tempo of MainStage. The line of best fit evaluation window is set to only include machine onsets at the new tempo.

4.2.4.5 Resetting

When MainStage stops playing, it sends a `stop` MIDI message to BeatSync. BeatSync then resets the MBD, deleting all recorded onsets. When MainStage begins playing again, the MBD begins detecting recording onsets and estimating the BP function as before.

4.2.5 Controller

The Controller compares the machine BP function $\theta_m(t)$ to the input BP function $\theta_i(t)$ and calculates MainStage's new tempo ω_m^{new} so that it becomes in sync with the input.

4.2.5.1 Design Constraints

In section 3.2.4.1, the problem of synchronising a DAW to a live drummer was compared to the problem of getting a car to drive alongside another car. MainStage is in fact far more ideal than a car in that it adopts a new tempo immediately; it doesn't need time to "get up to speed" like a car. Thus, the Controller is able to change MainStage's tempo to intercept the input BP function in a straight line, as shown in Figure 4.9.

One design constraint that must be considered is the fact that MainStage's audio is glitchy when its tempo changes too often. We therefore choose to include a minimum tempo change interval T_{tempo}^{min} as a design parameter. T_{tempo}^{min} is chosen by trial and error. The inclusion of a minimum tempo change interval means that there are discrete moments in time when BeatSync can make a tempo change. This can be thought of as a fixed Controller *sampling frequency*.

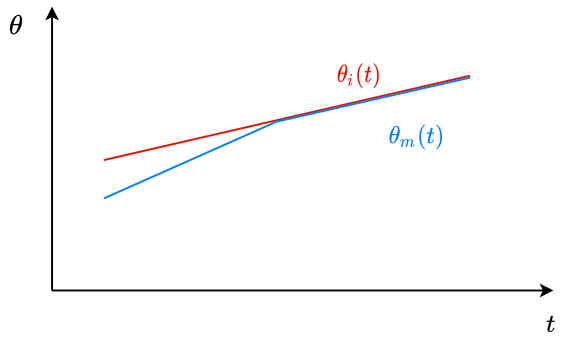


Figure 4.9: Machine BP intercepting input BP in a straight line.

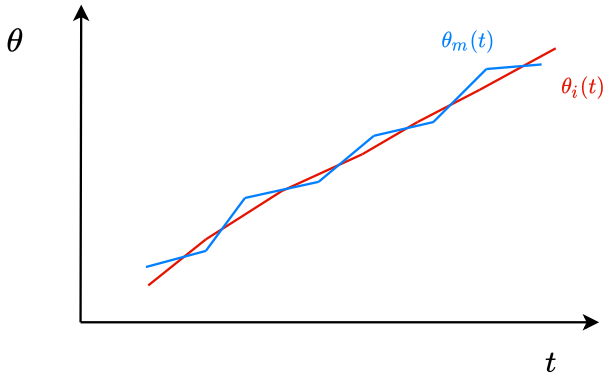


Figure 4.10: Oscillatory machine BP behaviour.

4.2.5.2 Controller Stability

While MainStage is ideal in its behaviour, the input is not. A drummer's tempo is constantly fluctuating. As a result, setting MainStage's tempo too radically can cause it to overshoot the input. This leads to oscillatory behaviour, as seen in Figure 4.10.

Therefore, one other design parameter $\triangle\omega_{\max}$ is selected for Controller stability. $\triangle\omega_{\max}$ constrains the new machine tempo value with the following condition: $\omega_i - \triangle\omega_{\max} \leq \omega_m^{new} \leq \omega_i + \triangle\omega_{\max}$

4.2.5.3 Synchronising to Bar and Beat

It has been noted that it is desirable for MainStage to be synchronised to the both the bar **and** the beat of the input. In other words, the "1" in MainStage's count of "**1** - 2 - 3 - 4 - **1** - 2 - 3 - 4 - **1** - 2 - ..." must occur at the same moment that the input (drummer)

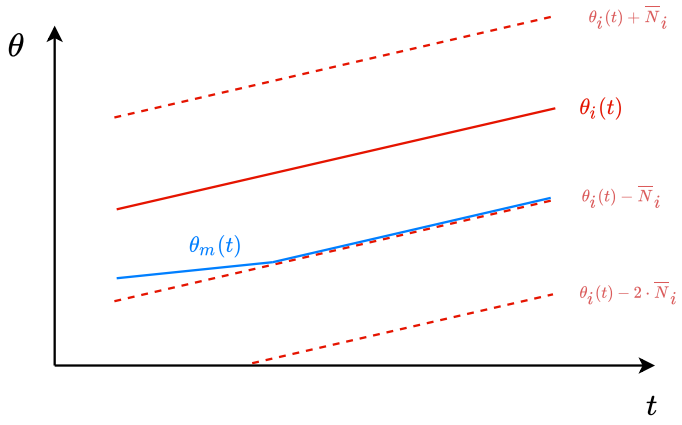


Figure 4.11: Family of input BP functions.

counts "1". It is not okay for MainStage to be at "2" while the input is at "1", even if they are aligned in beat.

In some cases, MainStage will start at the same time as the drummer and so synchronisation to the beat is also synchronisation to the bar. But, what about the case where the drummer hits play on MainStage after he starts playing, say two bars later[3]? Then the machine BP will be eight beats[4] behind the input BP. If that is the case, we don't want MainStage to catch up the entire eight beats. Rather, since the machine **Bar Beat Postion** is approximately (but most likely not exactly) the same as the input Bar Beat Position, the Controller must simply synchronise the machine to the nearest input beat.

A way of visualising this is by imagining a family of BP functions parallel to the input BP function but shifted in BP by a multiple of $\overline{N}_i$ (see Figure 4.11. The machine must synchronise to the BP function it is closest to.

Bearing this in mind, the Controller includes a step to symbolically shift the machine BP to be within $\frac{\overline{N}_i}{2}$ of the input (i.e. to "catch up" the eight beats). After the symbolic shift, Bar Beat Position can be ignored. The task of the Controller algorithm is simply to calculate MainStage's new tempo so that the machine BP function will intercept the input BP function.

[3]For the purpose of testing, the program has been set up to make MainStage start playing at the same time as the drummer. However it can easily be adapted so that if desired the drummer can choose when MainStage should come in.

[4]Assuming $\overline{N}_i = 4$. Then $N_i \times [2 \text{ bars}] = 8$ [beats].

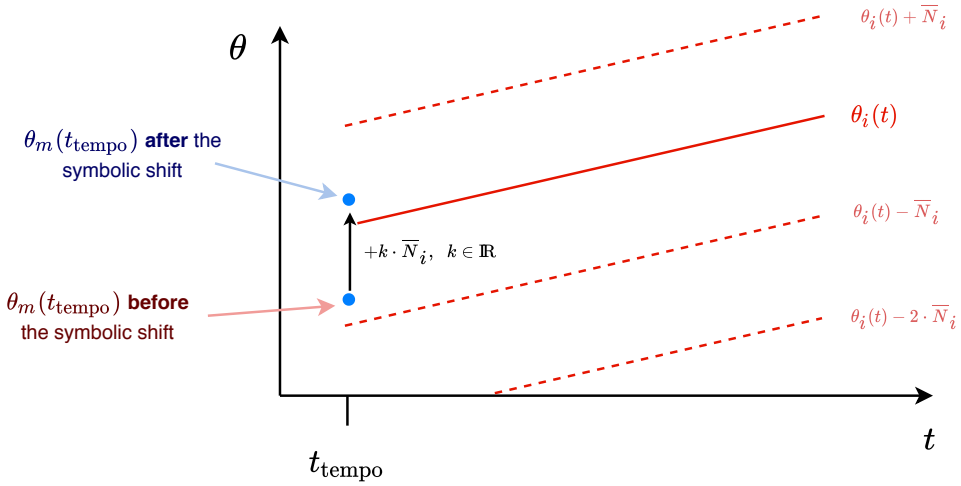


Figure 4.12: Controller symbolic machine Beat Position shift.

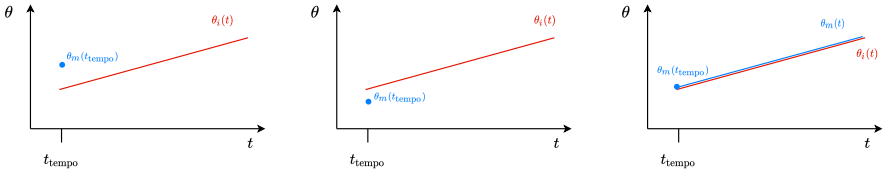


Figure 4.13: Controller checks if machine is ahead or behind or within acceptable error.

4.2.5.4 Algorithm

For a tempo change to be made at time t_{tempo} :

1. Symbolically shift machine BP to be within $\frac{\overline{N}_i}{2}$ of the input BP (see Figure 4.12):
   (a) Calculate error in Bar Beat Position: $e_{\overline{\theta}} = \overline{\theta}_m(t_{\text{tempo}}) - \overline{\theta}_i(t_{\text{tempo}})$
   (b) if $0 < e_{\overline{\theta}} < \frac{\overline{N}_i}{2}$ or $-\frac{\overline{N}_i}{2} \leq e_{\overline{\theta}} < 0$:
      i. $c_m \leftarrow \theta_i(t_{\text{tempo}}) + e_{\overline{\theta}} - \omega_m \cdot t_{\text{tempo}}$
   (c) else if $e_{\overline{\theta}} \geq \frac{\overline{N}_i}{2}$:
      i. $c_m \leftarrow \theta_i(t_{\text{tempo}}) + e_{\overline{\theta}} - \overline{N}_i - \omega_m \cdot t_{\text{tempo}}$
   (d) otherwise $e_{\overline{\theta}} < \frac{\overline{N}_i}{2}$:
      i. $c_m \leftarrow \theta_i(t_{\text{tempo}}) + e_{\overline{\theta}} + \overline{N}_i - \omega_m \cdot t_{\text{tempo}}$
2. Check if machine is ahead or behind or within acceptable error (see Figure 4.13):
   (a) Calculate BP error: $e_\theta = \theta_m(t_{\text{tempo}}) - \theta_i(t_{\text{tempo}})$

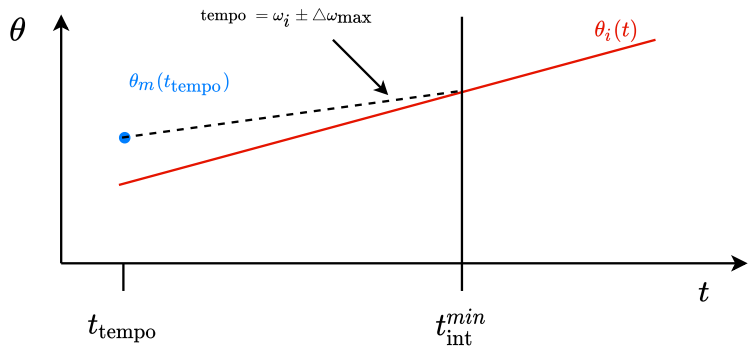


Figure 4.14: Controller calculates minimum interception time.

(b) If $|e_\theta| \leq \omega_i \cdot \epsilon_t \rightarrow$ machine is close enough to input beat, therefore match input tempo:

i. $\omega_m^{new} = \omega_i$

ii. End algorithm.

A time difference of ϵ_t occurs when the BP difference is $\omega_i \cdot \epsilon_t$. The acceptable time error is set at $\epsilon_t = 0.01$ s because delays under 0.01 s are unnoticable [5] and greater synchronisity is outside the tolerances of the system.

(c) Else if $e_\theta > 0 \rightarrow$ machine is ahead. Machine tempo must change to be slower than input.

i. $\triangle\omega_{\max} \leftarrow -\triangle\omega_{\max}$ (becomes negative)

(d) Otherwise, machine is behind . Machine tempo must change to be faster than input:

i. $\triangle\omega_{\max} \leftarrow \triangle\omega_{\max}$ (remains positive)

3. Find the minimum interception time given control parameter $\triangle\omega_{\max}$ (see Figure 4.14):

(a) The minimum interception time t_{int}^{min} is the time at the interception point of the input BP function $\theta_i(t) = \omega_i \cdot t + c_i$ and a line with tempo $\omega_i + \triangle\omega_{\max}$ through the point $(t_{\text{tempo}}, \theta_m(t_{\text{tempo}}))$:

$$t_{\text{int}}^{min} = \frac{1}{\triangle\omega_{\max}}[c_i - c_m + (\omega_i - \omega_m + \triangle\omega_{\max}) \cdot t_{\text{tempo}}] + c_m$$

4. Then the interception time t_{int} is the tempo change time soonest after t_{int}^{min} (see Figure 4.15):

(a) The interception point must be at a time when the machine's tempo is allowed to be changed again match the input tempo:

[5] http://whirlwindusa.com/support/tech-articles/opening-pandoras-box/

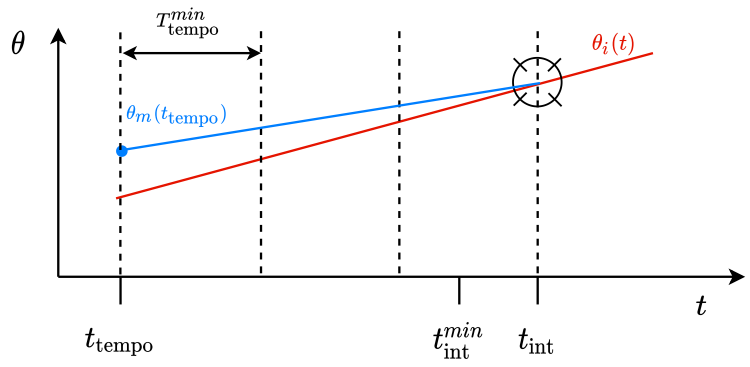


Figure 4.15: Controller calculates next possible interception time given fixed controller *sampling frequency*.

$$t_{\text{int}} = t_{\text{tempo}} + \text{ceil}\{\frac{t^{min}_{\text{int}} - t_{\text{tempo}}}{T^{min}_{\text{tempo}}}\} \cdot T^{min}_{\text{tempo}}$$

where ceil$\{x\}$ gives x rounded up to the nearest integer (e.g. ceil$\{2.1\} = 3$)

5. Finally, calculate the new machine tempo to be set at time t_{tempo} :

   (a) The new tempo to be set is simply the gradient of a BP function from the current machine BP point $(t_{\text{tempo}}, \theta_m(t_{\text{tempo}}))$ to the interception point $(t_{\text{int}}, \theta_i(t_{\text{int}}))$:

$$\omega^{new}_m = \frac{\theta_i(t_{\text{int}}) - \theta_m(t_{\text{tempo}})}{t_{\text{int}} - t_{\text{tempo}}} = \frac{\omega_i \cdot t_{\text{int}} - c_i - \omega_m \cdot t_{\text{tempo}} - c_m}{t_{\text{int}} - t_{\text{tempo}}}$$

4.2.5.5 Choice of Controller Parameter Values

Controller parameter values were chosen by trial and error to be the following:

- $T^{min}_{\text{tempo}} = 0.3$
- $\triangle\omega_{\text{max}} = 0.4$

4.2.6 Transport: A Container for the System

Each subsystem described so far (IBD, MBD and Controller) has been implemented by coding a seperate Python class (with the exception of the Onset Detector which is programmed in PureData). Having seperate classes allows each subsystem to be treated as a kind of black box.

A container program is required to interact with the three subsystems, and perform the tasks of:

- Setting up MIDI ports
- Receiving and timestamping MIDI messages from the Onset Detector
- Receiving and timestamping MIDI messages from MainStage
- Sending tempo changes and start messages to MainStage.

This container is called the Transport[6].

4.2.6.1 Keeping Track of Time

MIDI messages are timestamped as soon as they are received by calling the Python function `default_timer` from the `timeit` module. The `default_timer` function returns the time in seconds since the Python program began running. The function is also used to check if it is time for the Controller to "sample" the machine and input BP functions.

4.2.6.2 Transport Loop

The Transport runs in a loop, that does the following repeatedly:

- Checks if an input onset has arrived from the Onset Detector MIDI port.
  - If there is an onset: timestamp and send to the IBD.
- Checks if a machine onset has arrived from MainStage MIDI port.
  - If there is an onset: timestamp and send to the MBD.
- Checks if it is time for the next tempo change to happen.
  - If it is time: ask Controller for the new tempo MainStage must change to and send tempo change to MainStage.
  - Schedule the next tempo change time: $t^{next}_{\text{tempo}} = [\text{current time}] + T^{min}_{\text{tempo}}$.

[6]The name Transport is used to describe the time-keeping controls in most DAWs

4.2.6.3 Sending Tempo Changes

Tempo changes are sent to MainStage via a 14-bit MIDI *pitch bend* message.

In MainStage, a MIDI mapping is created to map pitch wheel messages linearly to MainStage's tempo, such that:

- a pitch bend value of 0 changes the tempo to 30 bpm.
- a pitch bend value of 16383 changes the tempo to 357.66 bpm.

  Steps to set up MainStage are included in Appendix.

The tempo range 30-357.66 bpm is chosen because it covers all foreseeable cases, and allows an even tempo resolution of 0.02 bpm[7]. In BeatSync, a tempo change value in beats per second is then converted to a pitch bend value as follows:

$$\text{pitch bend value} = \text{int}\{60 \times \frac{\omega_m^{new} - \omega_m^{min}}{\omega_{\text{res}}}\}$$

where

- ω_m^{new} is the new tempo given by the Controller in beats per second.
- ω_m^{min} is the minimum tempo in beats per second: $\omega_m^{min} = \frac{30 \text{ bpm}}{60 \text{ s}} = 0.5$ bps
- ω_{res} is the tempo resolution in beats per second: $\omega_{\text{res}} = \frac{0.02 \text{ bpm}}{60 \text{ s}} = \frac{1}{3000}$ bps

4.2.6.4 Starting MainStage

When BeatSync is initialised, it waits for a count-in from the drummer. After the final onset of the count-in (e.g. the fourth onset if $\overline{N}_i = 4$), the Transport asks the Input Beat Detector for the predicted time of the next beat. The Transport then does two operations:

1. Changes MainStage's tempo to match the input tempo: $\omega_m^{new} = \omega_i$
2. Sends a `start` MIDI message to MainStage at the predicted next input beat time.

[7]Calculation: $\frac{357.66-30}{2^{14}-1} = 0.02$

4.2.6.5 Compensating for Delays and Non-Ideal Effects

The following is a list of non-ideal effects experienced by BeatSync and how the Transport deals with them:

1. Onset Detector latency:
   (a) There is delay of approximately 20 ms* between a drum being hit and the Transport receiving the detected onset from the Onset Detector.
      i. Solution: Transport timestamps onsets as t_{ons} = [current time] − 20 ms before sending to the Input Beat Detector.

   *Calculated by comparing MIDI note on times to audio waveform spike times in Logic.

2. Controller execution time:
   (a) In the worst case, the Controller takes up to 10 ms* to calculate a new tempo.
      i. Solution: the Controller samples the BP functions and calculates a new tempo ahead of time. Since the Controller only works with symbolic representations of the BP functions, it doesn't need to do the calculation strictly at t_{tempo}. Instead the Controller does the calculation at $t = t_{\text{tempo}} - 0.01$ s but uses the actual value t_{tempo} in its calculations.

   *Most of the time, the Controller algorithm executes much quicker (around 100 μs), but it occasionally has a long execution time due to operating system interruptions.

3. MainStage start delay:
   (a) MainStage has a delay (25 ms) between receiving a `start` message and playing its first beat.
      i. Solution: Transport sends the start message 25 ms early.

4.2.6.6 Minimising Execution Times

BeatSync runs in a single thread, in a single loop. As a result, processes further down the loop must wait for processes higher up to finish processing before they can run. An example of where this a problem is if a MIDI note message is sent by MainStage while the Input Beat Detector is still processing an input onset. The Transport will only have a

chance to timestamp the MIDI note after the Input Beat Detector finishes its calculations. It is therefore crucial that execution times be as short as possible.

To achieve low execution times, timing tests were conducted to find the fastest method of doing various operations in python. One such test sought to find the quickest way to calculate the mean of an array of values. It was found that the `numpy.mean` function was much faster than the `statistics.mean` function.

While care was taken to minimise execution times, some finite execution time is unavoidable. Usually execution times are short enough that it is not a problem. And when there are timing delays they aren't a big issue becasue they are smoothed out by the line of best fit calculations done by the IBD and MBD.

Chapter 5

System Evaluation

The goal of this project was to create a system that synchronises a DAW to a live band with a drummer. The only way to truly validate whether or not this goal has been achieved is to get a real band with a real drummer to play with the system. The band would be able to say "Yes, this works and sounds great" or "No this needs improving". Such a qualitative test is crucial and will be included[1] in the evaluation of BeatSync.

While the system's final performance must be evaluated, it was also necessary to do preliminary testing in order to choose optimal design parameters such as $\triangle\omega_{\max}$. There are two problems with using a real live band for testing:

1. **Repeatability**

   A live band cannot repeat the exact same performance twice. Being able to repeat a test precisely is necessary to be able observe the effect of changing a design parameter.

2. **Long test hours**

   It is impractical to have a real band and drummer present for testing. It must be possible to test the system at any time of the day for as long as is necessary.

Given these two considerations, it was deemed necessary to devise the following:

1. **Automatic test procedure**: a method to test the system with a repeatable input.

[1] The qualitative test is conducted by myself since no band was available for this project.

2. **Quantitative evaluation method**: an evaluation method and metrics to quantify how well the system performs.

   (a) Performance metrics, though not perfect, give a good idea of how well the system performs without having to have someone listening to each test.

This chapter describes the design of the quantitative evaluation method and the automatic test procedure. Following that, final test results are given and a qualitative evaulation is presented.

5.1 Evaluation Method

5.1.1 Median Errors

Davies describes a quantitative evaluation method for beat trackers whereby the predicted beat times are compared to those annotated by a trained musician [10]. The trained musician taps in time to the input musical signal and the tap times are recorded. The taps are converted an audible sound (e.g. a cowbell sound) and where necessary, the trained musician manually corrects the tap times so that they are perceptually in time with the input audio. The beat times chosen by the trained musician are considered *ground truth*. This same method of producing ground truth beat times is used in [9, 3, 29, 23, 13]. The beat tracker's performance is then measured by looking at the median error of beat times.

This method of comparing to "ground truth" beat times has been widely adopted because it allows beat tracking systems to be compared to one another objectively. The trained musician is taken to be an ideal judge of beat position.

While these measures are perhaps well suited to testing offline beat trackers, Davies' proposed method of evaluation was found to be less suitable for the evaluation of BeatSync. It was found that median error values didn't always reflect whether or not MainStage **sounded** in sync. The median errors method will still be included in this system evaluation, but it will be validated by means of a listening test.

5.1.2 Listening Test

The most important tool of analysis in this project is audible feedback. If MainStage sounds in sync, then BeatSync is performing desirably. Thus, it was decided that alongside each formal quantitative test, a listening test would be performed.

The listening test was conducted by ear. In each listening test, both the audio input (drum recording) and MainStage's click could be heard. Whenever MainStage's click sounded out of sync with the drummer, that beat was considered inaccurately synchronised. It was also noted whether or not MainStage began in sync or took a number of beats to become in sync.

5.1.3 Test Performance Recordings

A set of recorded drum performances were collected as test data for the system. For each test performance, a seperate channel of audio was captured for kick and snare. Most of the recordings were recorded in isolation (without a band). In addition to the isolated recordings, six recordings from a live band were available, recorded prior to the project.

Each recording begins with a bar long count-in.

5.1.3.1 Isolated Drum Recordings

Several drum performances were recorded using the recording setup described in section 4.2.2. The drum patterns ranged from basic to complex and were played across a variety of tempos (80-190 bpm). The ground truth input beat times were produced by tapping in time to the test performance in Logic. The taps were recorded, converted to a MIDI file and, where necessary, adjusted to be perceptually in time with the drum recording.

5.1.3.2 Live Band Drum Recordings

Prior to the project, six live band performances with drums were recorded. Two of these were provided courtesy of Common Ground church, recorded by Dylan Kulenthal. In each live recording, the kick and snare were mic'd and recorded separately. Unlike with

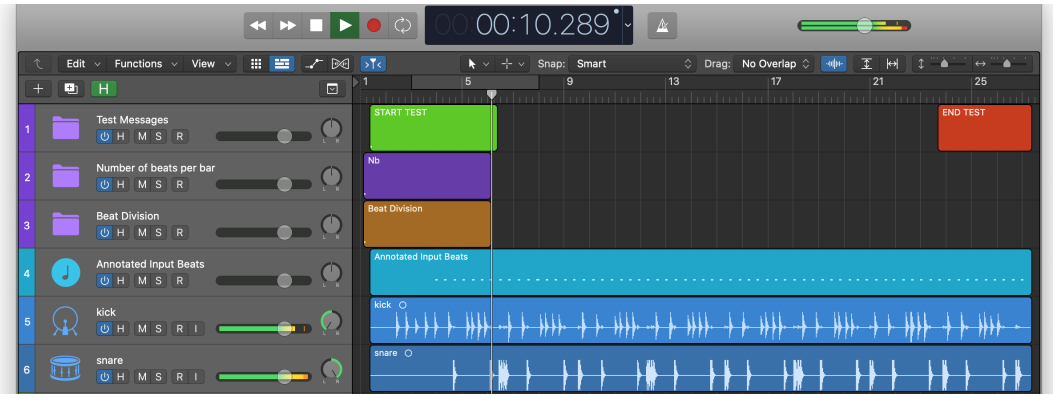


Figure 5.1: Logic automatic test rig.

the isolated drums which were recorded with contact mics, the live drums were recorded with dynamic mics. The rest of the band can be heard through the kick and snare mic's but are never loud enough to cause a false onset to be detected. This is evidence that both contact and dynamic mics are suitable input for the Onset Detector. The ground truth beat times were annotated as done with the isolated drum recordings.

5.1.4 Automatic Test Rig Design

An automatic test rig was designed using a combination of Logic and Python so that tests could be timed and repeated without human interferance. The Logic test rig setup for a single test performance is shown in Figure 5.1.

The Logic test rig sends a MIDI start message to the BeatSync Transport to begin the test and a MIDI stop message to end the test. The values of $\overline{N}_i$ and λ_i for each test are also communicated via MIDI. The kick and snare audio signals are played through the Onset Detector in real-time as if they were playing live. Figure 5.2 shows the signal flow between Logic, BeatSync, and MainStage.

The BeatSync Transport controls MainStage to keep in sync with the input and records the following:

- MainStage beat times
  - Occur at integer values of beat position. Given the chosen machine beat division $\lambda_m = 2$ this means that every second MIDI `note on` message received from MainStage is a beat, beginning with the first message.

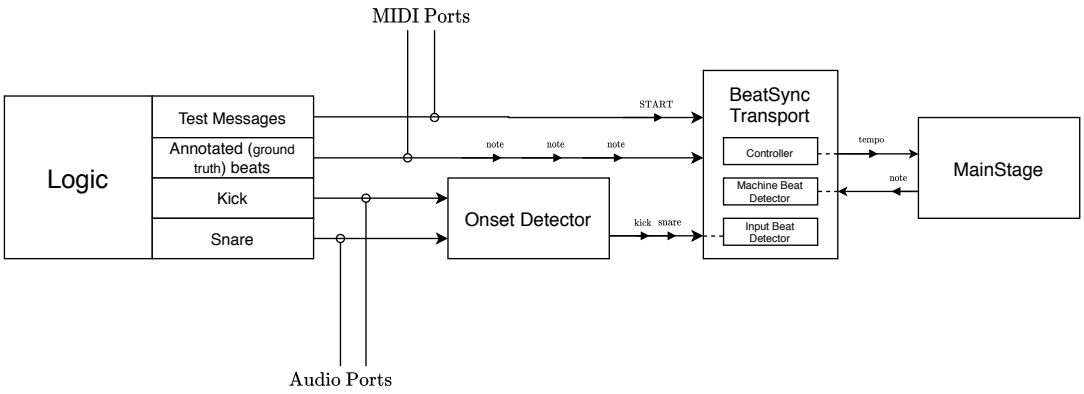


Figure 5.2: Automatic Test Rig signal flow.

- Input Beat Detector predicted beat times
- Annotated (ground truth) input beat times
  - Received via MIDI from the Logic test rig.

At the end of each test performance, the following results are calculated:

- Median of errors in MainStage's beat times
  - Each error in MainStage's beat time is calculated as the difference between MainStage's beat time and the corresponding annotated beat time.
  - This is the most important statistic as it indicates the performance of the whole system.
- Median of errors in IBD's predicted beat times.
  - For each predicition made (one for each onset), the error between the predicted beat time and the annotated beat time is calculated. Since there are often multiple onsets between two beats, there are often multiple predictions made for the same beat.
- Median of errors in the Controller's measured Beat Position difference.
  - Every time the controller makes a tempo change, it records the error in Beat Position between the machine and input
  - $error = \theta_m(t_{\text{tempo}}) - \theta_i(t_{\text{tempo}})$

These metrics allow the whole system performance to be tested, as well as the performance of the Input Beat Detector and Controller. Capturing the performance of the Controller and Input Beat Detector is useful in the case that BeatSync loses the beat. It makes it easy to see which subsystem is responsible for BeatSync missing the beat. For example, if MainStage sounds out of sync, and the controller median error is low but the IBD median error is high, it is clear that the IBD is to blame and needs improving.

5.2 Results

5.2.1 Table Of Results

The results for the entire set of tests is presented in Table 5.1. Tests where BeatSync lost the beat are in bold. Tests on live band recordings are below the bold horizontal line (tests 26-31).

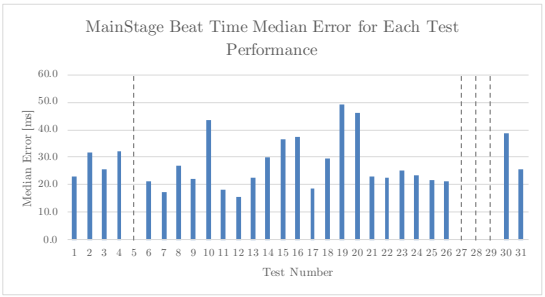
5.2.2 Whole System Performance

The performance of the whole BeatSync system was determined by comparing MainStage beat times to the annotated (ground truth) beat times. This was done in two ways:

1. Median Error:
   (a) The median of the errors between MainStage's beat times and the annotated beat times was recorded.
   (b) The median error value for each test performance is shown in column four of Table 5.1, and plotted in Figure 5.3.
2. Accuracy:
   (a) The accuracy of MainStage's beat times was measured as the percentage of input beats accurately synchronised.
   (b) Whether or not a beat was accurately synchronised was determined by the Listening Test.
   (c) The accuracy values are recorded in column eight of Table 5.1.

Table 5.1: System Evaluation results.

Test Information			Median Errors			Listening Test		
Test Number	**Test Length [beats]**	**Mean Tempo [bpm]**	**MainStage Beat Median Error [ms]**	**IBD Median Error [ms]**	**Controller Median Error [beats]**	**Number of beats to steady-state**	**Number of beats accurately synchronised**	**Remained in sync for full performance?**
1	65	83.5	22.8	21.1	0.0156	0	65 (100.0%)	Yes
2	65	105.3	31.8	25.8	0.0195	0	65 (100.0%)	Yes
3	98	125.9	25.7	19.0	0.0205	0	98 (100.0%)	Yes
4	96	144.5	32.3	27.5	0.0263	0	96 (100.0%)	Yes
5	**82**	**157.5**	**3168.2**	**4055.6**	**0.0544**	**-**	**0 (0%)**	**No**
6	129	87.7	21.0	23.1	0.0118	0	126 (97.7%)	Yes
7	113	106.3	17.2	17.3	0.0137	6	107 (94.7%)	Yes
8	145	119.1	27.1	24.7	0.0181	0	145 (100.0%)	Yes
9	129	145.4	22.3	21.9	0.0180	0	129 (100.0%)	Yes
10	129	157.3	43.6	37.8	0.0398	6	123 (95.3%)	Yes
11	129	86.6	18.2	18.8	0.0112	0	129 (100.0%)	Yes
12	113	103.3	15.5	13.4	0.0122	4	109 (96.5%)	Yes
13	81	124.0	22.5	23.8	0.0173	4	77 (95.1%)	Yes
14	114	138.0	29.8	24.9	0.0184	8	106 (93.0%)	Yes
15	129	158.6	36.6	29.0	0.0341	10	119 (92.2%)	Yes
16	129	85.5	37.3	39.4	0.0107	0	129 (100.0%)	Yes
17	129	101.6	18.6	15.4	0.0137	8	115 (89.1%)	Yes
18	129	116.3	29.6	29.9	0.0156	8	118 (91.5%)	Yes
19	129	142.8	49.3	46.8	0.0282	8	115 (89.1%)	Yes
20	129	160.1	46.3	44.9	0.0349	8	118 (91.5%)	Yes
21	65	80.5	23.0	22.6	0.0113	0	64 (98.5%)	Yes
22	65	94.2	22.4	23.6	0.0144	8	55 (84.6%)	Yes
23	145	109.9	25.3	26.0	0.0163	8	137 (94.5%)	Yes
24	129	135.7	23.2	20.6	0.0232	8	121 (93.8%)	Yes
25	129	188.6	21.6	20.3	0.0388	0	129 (100.0%)	Yes
26	494	107.5	21.2	22.6	0.0202	0	489 (99.0%)	Yes
27	**642**	**141.4**	**193.9**	**373.4**	**0.0282**	**0**	**50 (7.8%)**	**No**
28	**190**	**69.9**	**150.8**	**290.0**	**0.0157**	**0**	**77 (40.5%)**	**No**
29	**58**	**69.5**	**666.5**	**110.0**	**0.0174**	**0**	**10 (17.2%)**	**No**
30	50	76.9	38.7	38.0	0.0153	4	44 (88.0%)	Yes
31	143	73.9	25.7	21.2	0.0100	0	141 (98.6%)	Yes



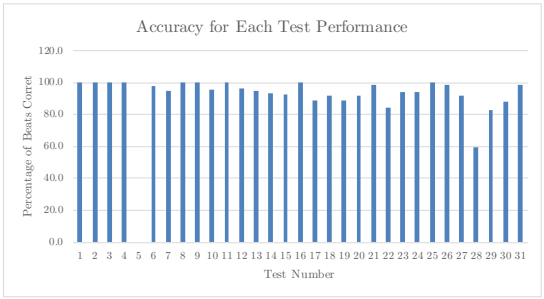


Figure 5.3: MainStage beat time median error (left) and accuracy (right) for each test performance. Cases where BeatSync lost the beat are excluded from the median error graph. They are indicated by dotted lines.

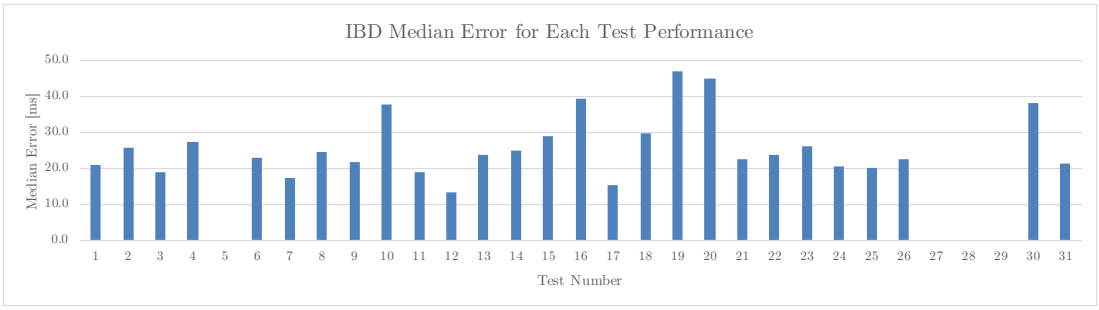


Figure 5.4: IBD median error for each test performance. Cases where BeatSync lost the beat are excluded from the plot (indicated by a blank space.)

The median error and accuracy for each performance is plotted in Figure 5.3

Of the 31 test performances, 27 were syncrhonised accurately to the end. This means that BeatSync kept MainStage in sync for the entire performance. In three cases (tests 27, 28, 29), MainStage began in sync, but lost the beat at some point during the performance. In one case (test 5), MainStage immediately went out of sync and never recovered.

5.2.3 Input Beat Detector Performance

The performance of the Input Beat Detector was measured by calculating the median value of the errors in the IBD's predicted beat times (compared to the ground truth annotated beat times). The IBD median error for each test performance is shown in Figure 5.4.

The IBD median error plot has a strong correlation to the MainStage median error plot. This can easily be seen by viewing the plots side by side, as shown in Figure 5.5.

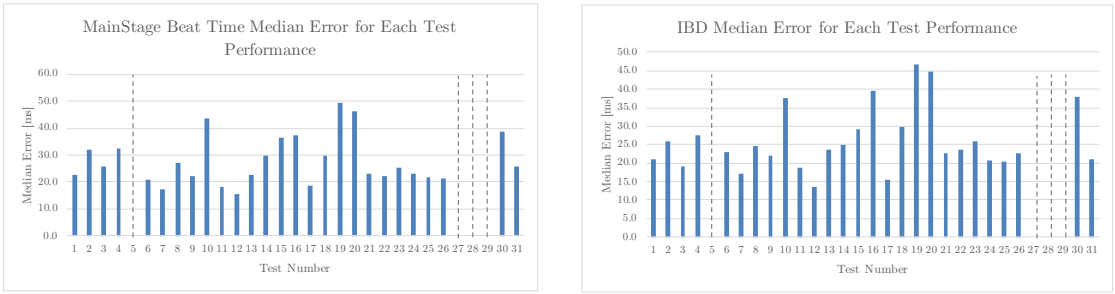


Figure 5.5: MainStage beat time median error (left) and IBD median error (right) for each test performance. Cases where BeatSync lost the beat are excluded from both plots, indicated by a dotted line.

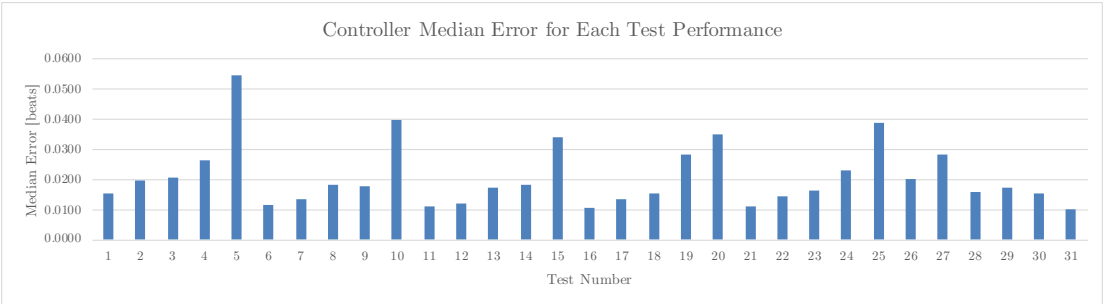


Figure 5.6: Controller median errors for each test performance.

The strong correlation between IBD median error and MainStage beat time median error suggests that the controller accurately tracks the Beat Position function estimated by the IBD.

5.2.4 Controller Performance

The Controller's performance is measured as the median of errors (or differences) in the machine and input BP functions, measured in beats. The median of errors gives an indication of how well the Controller is able to make MainStage's BP function be aligned to the input BP function. In other words, it indicates how "close" MainStage is to the input predicted by the IBD. The plot of median errors for each test performance is shown in Figure 5.6.

The Controller median error seems to have a repeating pattern in multiples of 5 of the test number. This is most likely due to that the fact that when the test performances were recorded, a drum pattern was repeated five times at increasing tempos. The correlation between tempo and Controller median error can more easily be seen in a plot of Controller median error vs. performance tempo, as shown in Figure 5.7.

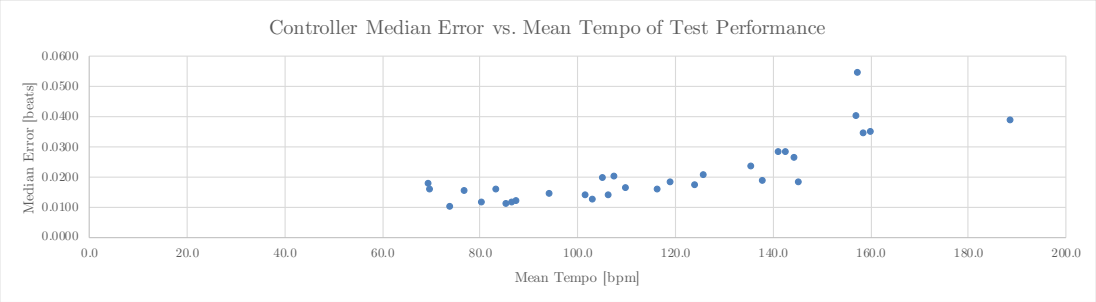


Figure 5.7: Controller median errors for each test performance.

Figure 5.7 suggests that the Controller performs better at lower tempos, but not by much. In the case of all the performances, including the performances where BeatSync lost the beat, the Controller median error is less than 0.06 beats. A BP difference of 0.06 beats at 180 bpm equates to a 20 ms time delay between MainStage and the input BP function. A delay of 20 ms is imperceptible to most people and will sound in sync. It can therefore be concluded that the controller performed adequately well for all test performances.

5.3 Discussion

In this section, the system evaluation results are interpreted and discussed. The discussion is conducted by way of observing cases where BeatSync performed perfectly, cases where it was almost perfect with unstable starts and cases where BeatSync lost the beat. Finally, BeatSync's performance on the set of 5.1.3.2.

5.3.1 Perfect Performances

Nine of the test performances were perfectly synchronised by BeatSync. This means that in the listening test, all of MainStage's were considered to be in sync with the drum performance. BeatSync's perfromance in these cases is very satisfying. The MainStage beat time median error, IBD median error, and Controller median error plots are shown for a three of these performances in Figure .

Note in Figure 5.8 that there is a strong correlation between IBD error and MainStage beat time error. This confirms that the Controller is accurate in synchronising MainStage to the IBD predicted beat, whether correct or not. The plots make it seem like MainStage was slightly off beat occasionally. For example, in track 16 (middle, left plot) the MainStage

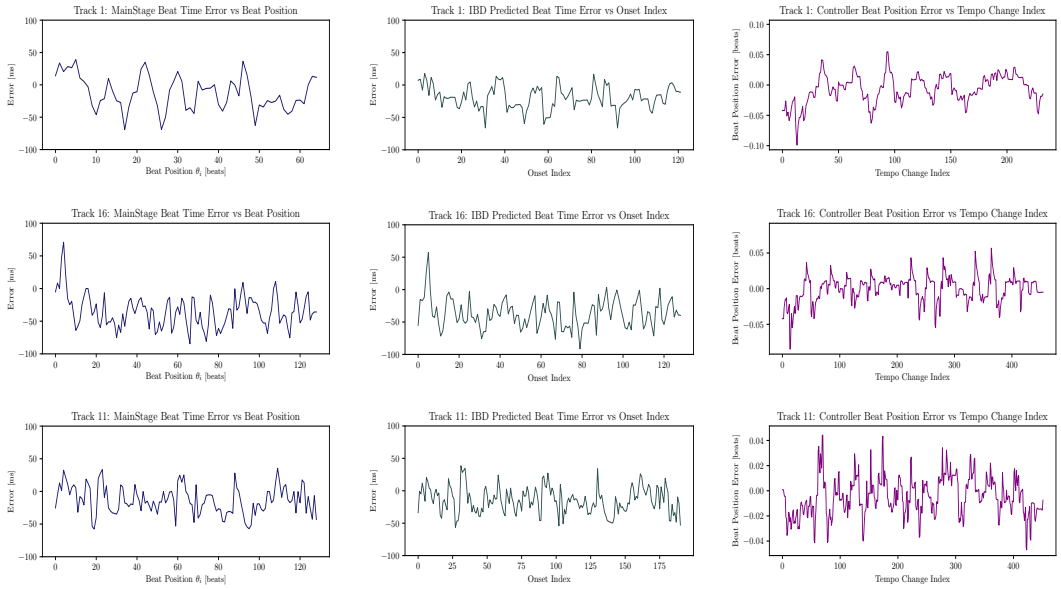


Figure 5.8: MainStage beat time error (left), IBD predicted beat time error (middle) and Controller beat position error (right) for three performances. The Controller BP error plots are normalised.

Beat Time Error plot suggest there is a 60 ms error between MainStage's beat and the annotated ground truth beat towards the beginning of the performance. In reality, the listening test confirmed that all beats sounded accurately synchronised. Possible reasons for noise in the error time recordings are as follows:

- Unknown latency in MIDI port connection.
- Operating system interrupts causing Python program execution to be delayed.
- Limited accuracy of the timeit module in Python.
  - The timing clock in Python is known to have a limited timing resolution. Perhaps this is reflected in the timer values recorded in these tests.
- Timing limitations of BeatSync's transport.
  - If a MainStage beat arrives at the same as an annotated beat from the Automatic Test Rig, one will be processed before the other meaning there is a delay between the timestamping of each MIDI message.

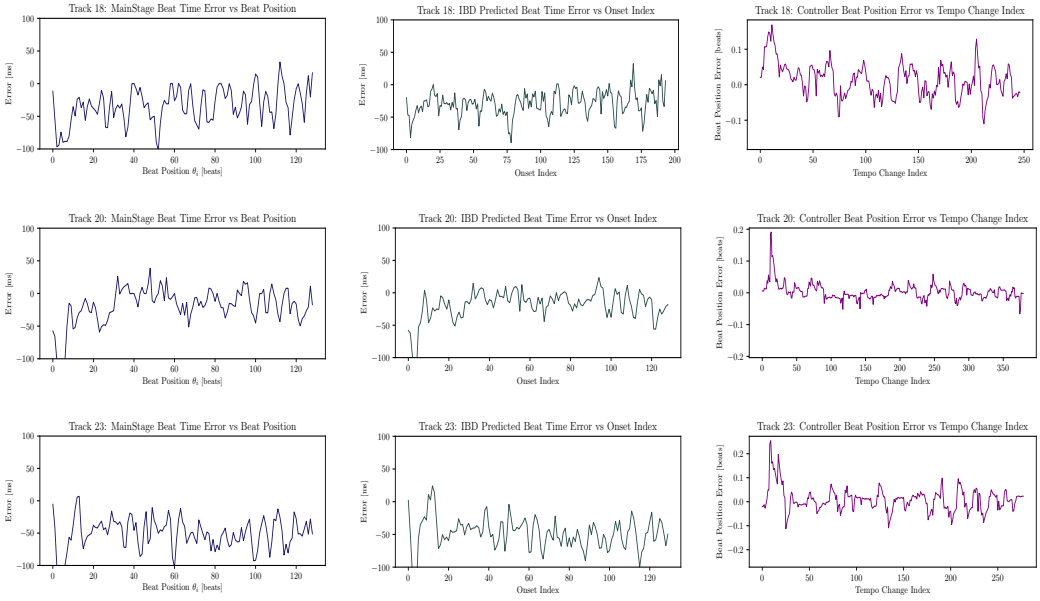


Figure 5.9: MainStage beat time error (left), IBD predicted beat time error (middle) and Controller beat position error (right) for three performances that started out of sync. The Controller BP error plots are normalised.

5.3.2 Performances with Unstable Starts

Some of the performances began out of sync but reached synchonisity after a few beats. This was the case for 13 of the performances. In all these cases, BeatSync started MainStage at too high a tempo and had to slow down MainStage down. The behaviour was clearly audible in the listening test. Figure 5.9 shows the error plots for three of these cases.

In Figure 5.9, it can be seen that the error plots confirm what was heard in the listening test. The IBD incorrecly predicted the input beat BP function from the bar long count-in at the start of the performance. This caused BeatSync to start MainStage at too high a tempo. The Input Beat Detector was able to regain accuracy in its predictions. In all Note that the IBD predicted beat time error is high at the start of the performances. This suggests that the IBD is to blame for the fast start.

There is no observable relationship among the performances that began out of sync. Nothing seems to correlate them. Perhaps the drummer simply slowed down after the count-in. If that is the case, it would be desirable for the IBD to be able to predict the count-in behaviour of the drummer. More investigation is required to determine the reason for the IBD predicting too high a tempo at the start. However, that is a topic for future study.

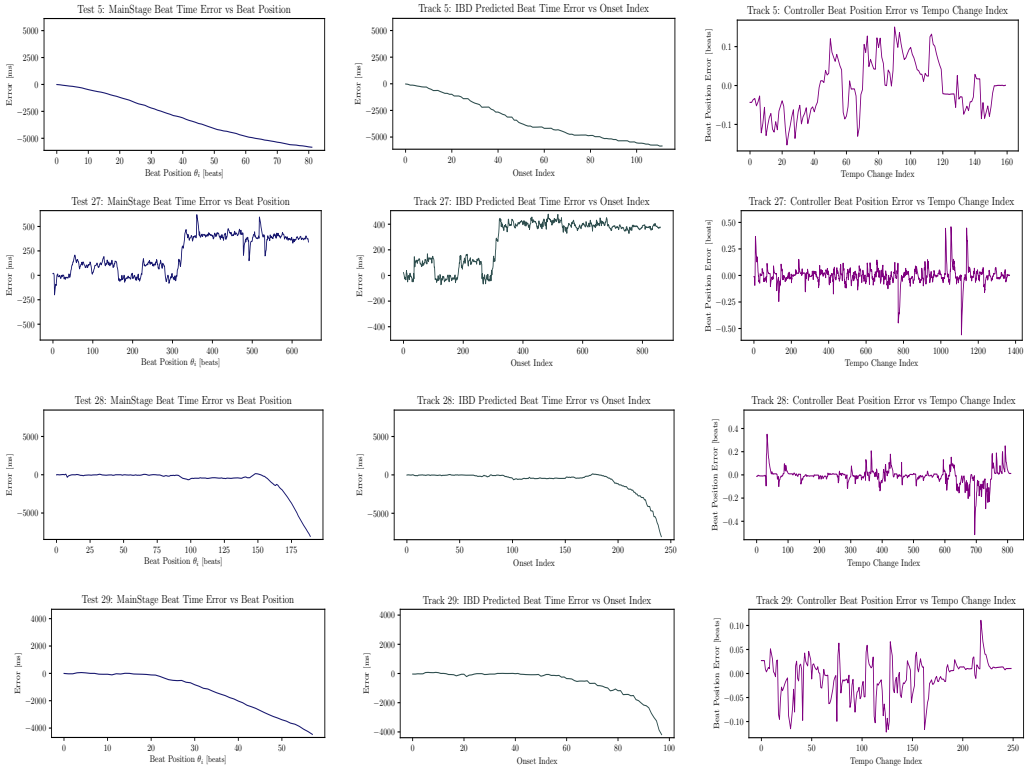


Figure 5.10: MainStage beat time error (left), IBD predicted beat time error (middle) and Controller beat position error (right) for three performances that started out of sync. The Controller BP error plots are normalised.

5.3.3 Inaccurately Synchronised Performances

Four of the performances were inadequately synchronised by BeatSync. Three of these began in sync but went out of sync later on. In one case BeatSync lost the beat immediately. The error plots for the four badly synchronised performances are shown in Figure 5.10.

In test 5, the IBD was unable to accurately predict the input beat from the count-in. This caused MainStage to become immediately out of sync and begin playing at a completely different tempo to the drummer. In the case of tests 28 and 29, the IBD accurately predicted the input beat for a portion of the performance. At some point in the performance however, the tempo changed rapidly and the IBD began incorrectly quantising incoming onsets (e.g. by saying that an onset at $\overline{\theta}_i = 2$ is at $\overline{\theta}_i = 1.75$). This caused the IBD's prediction of input BP to quickly become inaccurate and MainStage went out of sync with the drummer.

Test 27 had a surprising outcome. Unlike in the case of tests 5, 28 and 29, when MainStage

started to go out of sync, the IBD didn't completely lose the input beat. MainStage began playing at the same tempo as the input but one beat ahead. This can be seen by the horizontal lines in the error plots for test 27 in Figure 5.10. The test is still a fail, since MainStage playing one beat ahead is unacceptable, but it shows that the IBD has some hope of error correcting. If the IBD perhaps had some way of recognizing the repeating drum pattern being played, it would be able to see that it is one beat ahead and adjust its estimate by a whole beat.

A promising note is the fact that in all four cases where the IBD lost the beat, the Controller performed. The Controller did its job of synchronising MainStage to the beat predicted by the input beat detector, even if was the wrong beat.

Tests 27-29 reveal a limitation in the IBD's algorithm. The IBD is not well suited to track rapid tempo changes. By design, the IBD favours stability over responsiveness to tempo changes. The rest of the tests are evidence that this design is a good choice for some cases. But if the tempo change is too rapid, then the assumption of a constant tempo into the future is inadequate. The tempo plots for tests 27-29, given earlier in Figure 3.2 , show that the performances had far from constant tempos. Test 27's tempo is almost sinusoidal. Over a short period of time, the BP function of test 27 could perhaps best be represented by a parabolic curve. Further study is required to determine how best to describe the BP function of live performances.

All four of these tests indicate the need for the IBD to have some mechanism for error correction. As it stands, the IBD has no "awareness". It doesn't know when its lost the beat. Davies describes a beat tracker that has both a general state and a context-dependant state [10]. The context-dependant state predicts the location of beats based on its current estimate of beat position. The general state keeps no estimate, and instead continually makes new predictions based on the location of onsets. The general state is always looking for a new beat to track. By considering the general state, the beat-tracker is able to continually validate its hypothesis from the context-dependant state. The two state method could be used to improve the IBD by giving it the ability to know when it is out of sync.

5.3.4 Live Band Recordings

Three out of the six live band recordings were accurately synchronised. The three correctly synchronised tests show that BeatSync is a working proof of concept. It has an ability to

synchronise MainStage to a live drummer in a way that is pleasant to listen to. However, it cannot be considered robust for live performance, evidenced by the three incorrectly synchronised performances.

The biggest limitation of these results is the fact that they are limited in number. In future automatic synchronisation work, it would be desirable to be test the system on a large number of live recordings.

Chapter 6

Conclusions

In this project report, an automatic synchronisation system BeatSync was presented. The motivation to build the automatic synchronisation system was to provide an alternative to a click track for musicians that want to use a Digital Audio Workstation (DAW) in a live performance. As a starting point, the published automatic synchronisation system B-Keeper was tested and found to be a useful system. However, B-Keeper had a limitation in that it was unable to make a tempo change at a time other than when a drum was hit. As a result, B-Keeper would send a new tempo to the DAW everytime a drum was hit. The high frequency of tempo changes caused the DAW MainStage to exhibit undesirable audio artefacts.

B-Keeper's inability to limit the number of tempo changes made motivated the desire for a new system to built. A new system was proposed, called BeatSync, and was based on a high level understanding of musical structure, outlined in a Beat Theory Chapter. The Beat Theory Chapter explained how the perception of beat in music is due to an expectation that musical events will occur at some time in the future. BeatSync utilises this power of expectation. BeatSync maintains at all times an estimation of when beats are expected to occur. The estimate is stored in a symbolic representation called a Beat Position function.

The use of a symbolic representation in the prediction of beat locations is what distinguishes BeatSync from B-Keeper. The symbolic representation allows BeatSync to make tempo changes at any time, as frequently or infrequently as desired.

BeatSync was designed with modern rock and pop music in mind. To this end, the

system was evaluated on a set of drum recordings featuring patterns common in rock and pop music. BeatSync was successfully able to synchronise the Digital Audio Workstation MainStage to many of the test recordings. The synchronisation achieved was shown to satisfactory by means of a listening test. In a few cases, BeatSync lost the beat, causing MainStage to become out of sync with the drummer. These cases are evidence that further work is required to make BeatSync a viable tool for live performance. As it stands, BeatSync works in cases where the tempo fluctuations in a performance aren't too radical.

The project goal was achieved. A real-time automatic synchronisation system was built and shown to be useful. The successes of the system motivate the desire for future work to be done to improve the system.

Chapter 7

Recommendations

In the following Chapter, future work to improve the field of automatic synchronisation is presented.

7.1 Future Design Projects

The BeatSync system presented in this project report has already shown itself to be useful. However, much work is required before BeatSync can be used in a live performance without fear of it losing the beat. In order to improve BeatSync, the following design projects are recommended:

1. Onset detector for other instruments
    (a) Guitar
    (b) Voice
    (c) Piano
2. Beat-tracking study to improve the Input Beat Detector
    (a) Many approaches could be used to improve the Input Beat Detector. Some of these were discussed earlier in the Literature Review, and also proposed in the System Evaluation. Possible approaches are the following:
        i. Multiple hypotheses of tempo and beat position

ii. Pattern recognition

iii. Non-constant Beat Position functions

iv. Utilising harmonic information

3. Implement BeatSync as a standalone application

   (a) As it stands, BeatSync requires both a Python and PureData program to installed and correctly connected to work. This is unsuitable for easy use.

   (b) BeatSync could be implemented as an Audio Unit plugin in MainStage.

4. Implement BeatSync on other platforms

   (a) BeatSync could be adapted for use with other DAWs and other programming languages.

7.2 Future Design Tools

In this section tools are presented to aid future development of BeatSync. It was not possible to properly use these tools in this project due to time constraints. They are presented here as a reccomendation for future work in the field of automatic synchronisation.

7.2.1 Offline Test Procedure

A method to test the IBD offline was devised to allow the accuracy of the IBD to be tested without having to run the system in real-time. The benefit of running the algorithm offline is that multiple variations of the design parameters can be tested in a short amount of time. The algorithm still

Test files were produced as follows:

1. A set of test performances were recorded into Logic as described in section 5.1.3
2. Each test recording was then played through the Onset Detector in real-time.
3. The MIDI messages outputted by the Onset Detector were routed back into Logic where they were recorded and stored as a MIDI file.

Each test performance includes a bar-long count-in. Values of $\overline{N}_i$ and λ_i for each test performance are stored in meta messages in the test MIDI file.

Each recorded MIDI file is the exact data [1] that is sent to the IBD when BeatSync is run in real-time. The performance of the IBD for a chosen set of design parameters was then tested as follows:

1. Each `note_on` MIDI message is sent sequentially to the IBD.
2. For each message (onset) received, the IBD runs its algorithm and returns a predicted next beat time.
   (a) The IBD predicts the next beat time using its current estimate of the input BP function and equation 3.2.
3. The error between the predicted next beat time and a *ground-truth*[2] # beat time is recorded.

The above test procedure can be carried out for every test file and repeated for various choices of design parameters.

7.2.1.1 Offline Test Results

The median error can be computed for each iteration of design parameters and be used to decide the optimal choice.

7.2.1.2 Stochastic Drummer

A Stochastic Drummer program was written based on Robertson's description of a stochastic drum machine [28] .

The Stochastic Drummer simulates a drummer by playing a repeating pattern with three kinds of variability:

[1] One exception: MIDI files store time in ticks (there are 960 ticks per second) whereas the IBD accepts time in seconds. For the test procedure, the ticks are simply converted back into seconds.

[2] The method for determining the ground-truth beat times is explained in Section 5.1

1. Note probablility:

   Each note of the pattern has an associated probability that it will play.

   - Stochastic drummer decides if a note should play by generating a random number between 0 and 1. If the note probability (between 0 and 1 inclusive) is higher than the random number, the note will play.

2. Tempo variance:

   At each beat, the Stochastic Drummer adds to the inter-beat interval Gaussian noise of zero mean and a set standard deviation (0 to 16ms).

   - Example: for an initial tempo of 120 bpm, increasing the inter-beat interval by 16 ms results in a ˜4 bpm tempo decrease.

3. Onset time variance:

   At each beat or sub beat the Stochastic drummer adds Gaussian noise of zero mean and a set standard deviation (0 to 64ms) to the onset time error.

   - An onset time error standard deviation of 0 will result in onsets being played exactly on the perceptual beat or sub beat.
   - Onset time error standard deviation greater than 0 will result in onsets being played off the beat or sub beat by some amount.

The Stochastic Drummer program takes as input: initial tempo ω_{init}, beat division λ_i, bar length $\overline{N}_i$, number of bars, kick and snare pattern matrix, step probability list, inter-beat interval standard deviation and onset time standard deviation.

The Stochastic Drummer program produces as output:

- A MIDI file of kick and snare notes.
- A MIDI file of ground truth beats.
  - Each ground truth beat is annotated as a MIDI note message.

The output MIDI file of kick and snare notes is imported into Logic where each note is replaced with a kick or snare drum sample automatically. The drum samples are provided by Logic and are recorded with a dynamic or condensor microphone.

7.2.2 Programs Available Online

The Offline Test Procedure, Stochastic Drummer program, and all code used in this project can be found at this GitHub page. The GitHub page features the following tools and programs:

- Offline Test Procedure
- Stochastic Drummer
- B-Keeper Python Implementation
- MockStage and Simulator
  - Python programs developed for offline testing of B-Keeper
- All BeatSync files

Bibliography

[1] 2020. URL: https://en.wikipedia.org/wiki/Contact_microphone.

[2] *Ableton Live 10 Screen Shot*. URL: https://www.ableton.com/en/live/.

[3] Eric Dean Battenberg. "Techniques for machine understanding of live drum performances". PhD thesis. UC Berkeley, 2012.

[4] *Beat (music)*. URL: https://en.wikipedia.org/wiki/Beat_(music).

[5] John Bispham. "Rhythm in music: What is it? Who has it? And why?" In: *Music Perception*24.2(2006), pp. 125–134.

[6] Katylin Brink. *Genesis Conference with River Valley Worship*. Goose The Drummer. URL: https://goosethedrummer.com/.

[7] Griffin Brown. *Beat Making 101: How to Make a Beat*. 2020. URL: https://www.izotope.com/en/learn/beat-making-101-how-to-make-a-beat.html.

[8] Ivan Brunoand Paolo Nesi. "Automatic synchronisation based on beat tracking". In: *MAXIS 2003 Conference, Leeds (UK)*. 2003.

[9] Nick Collins. "Drumtrack: beat Induction from an acoustic Drum Kit with synchronised Scheduling." In: *ICMC*. Citeseer. 2005.

[10] Matthew EP Daviesand Mark D Plumbley. "Context-dependent beat tracking of musical audio". In: *IEEE Transactions on Audio, Speech, and Language Processing*15.3(2007), pp. 1009–1020.

[11] Musical Dictionary. *Beat*. Oct. 2018. URL: https://musicaldictionary.com/beat/.

[12] Simon Dixon. "Automatic extraction of tempo and beat from expressive performances". In: *Journal of New Music Research*30.1(2001), pp. 39–58.

[13] Masataka Goto. "An audio-based real-time beat tracking system for music with or without drum-sounds". In: *Journal of New Music Research*30.2(2001), pp. 159–171.

[14] Masataka Gotoand Yoichi Muraoka. “A beat tracking system for acoustic signals of music”. In: *Proceedings of the second ACM international conference on Multimedia*. 1994, pp. 365–372.

[15] Masataka Gotoand Yoichi Muraoka. “A real-time beat tracking system for audio signals”. In: *ICMC*. 1995.

[16] Masataka Gotoand Yoichi Muraoka. “Music understanding at the beat level: Real-time beat tracking for audio signals”. In: *Computational auditory scene analysis*(1998), pp. 157–176.

[17] Stephen Hainsworth. “Beat tracking and musical metre analysis”. In: *Signal processing methods for music transcription*. Springer, 2006, pp. 101–129.

[18] Peter Crossley Holland. *Rhythm*. URL: `https://www.britannica.com/art/rhythm-music`.

[19] JOHN R Iversen. “21 In the beginning was the beat: evolutionary origins of musical rhythm in humans”. In: *The Cambridge companion to percussion*(2016), pp. 281–295.

[20] Kristoffer Jensenand Tue Haste Andersen. “Beat estimation on the beat”. In: *2003 IEEE Workshop on Applications of Signal Processing to Audio and Acoustics (IEEE Cat. No. 03TH8684)*. IEEE. 2003, pp. 87–90.

[21] Edward W Large. “Beat tracking with a nonlinear oscillator”. In: *Working Notes of the IJCAI-95 Workshop on Artificial Intelligence and Music*. Vol. 24031. 1995.

[22] James A.H. Murray. *The Compact Edition of the Oxford English Dictionary II*. Oxfordand New York: Oxford University Press, 1971.

[23] Joao Lobato Oliveiraet al. “IBT: A real-time tempo and beat tracking system”. In: (2010).

[24] Miller S Puckette, Miller S Puckette Ucsd, Theodore Apel,et al. “Real-time audio analysis tools for Pd and MSP”. In: (1998).

[25] Sydney Castle Roberts. *Interpreting Rhythmic Structures Using Artificial Neural Networks*. 1996.

[26] Andrew Robertson. “Interactive real-time musical systems”. PhD thesis. 2009.

[27] Andrew Robertsonand Mark Plumbley. “B-Keeper: A beat-tracker for live performance”. In: *Proceedings of the 7th international conference on New interfaces for musical expression*. 2007, pp. 234–237.

[28] Andrew Robertsonand Mark D Plumbley. “Synchronizing sequencing software to a live drummer”. In: *Computer Music Journal*37.2(2013), pp. 46–60.

[29] Eric D Scheirer. “Tempo and beat analysis of acoustic musical signals”. In: *The Journal of the Acoustical Society of America*103.1(1998), pp. 588–601.

[30] *Ultimate Ears in ear monitors*. URL: `https://pro.ultimateears.com`.

Appendix A

Setting Up BeatSync

A.1 MainStage Setup

A.1.1 Machine Onsets Out

BeatSync learns MainStage's Beat Position by receiving MIDI note on messages via a dedicated MIDI port. Follow these steps to set:

1. Run `BeatSync.py`:
   (a) BeatSync must be on in standby mode in order for its virtual MIDI ports to be available.
2. Create a new channel strip in MainStage:
   (a) Use the "Add Channel Strip" button in MainStage and select the "External MIDI" channel type.
3. Change the MIDI Output of the channel strip to "BeatSync MainStage In".
4. Add a Scripter MIDI effect by clicking in the MIDI FX area.
5. Copy the following code into a new script:

```
var NeedsTimingInfo = true;
ResetParameterDefaults = true;
```

```
function ProcessMIDI() {
        var info = GetTimingInfo();
        var Nb = info.meterNumerator; // Num beats per bar
        if (info.playing) {
        // Calculate beat to schedule
        var beat_division = 2;
        var lookAheadEnd = info.blockEndBeat;
        var beatToSchedule = Math.ceil(info.blockStartBeat * beat_division) /
                                                        beat_division;

                while (beatToSchedule >= info.blockStartBeat &&
                                beatToSchedule < lookAheadEnd) {
                // Send note if it falls within this buffer
                // Find the BarSubBeatPosition
        // 1. Find the BarBeatPosition

                var BBP = (beatToSchedule-1)%Nb + 1
                                // Nb is the num. beats per bar
                // 2. Find the BarSubBeatPosition
                var BSBP = beat_division*(BBP-1) + 1

                        // Set the note pitch based on BSPB
                        var on = new NoteOn;
                        on.pitch = 59 + BSBP;
                        on.velocity = 127;
                        on.sendAtBeat(beatToSchedule)

                        var off = new NoteOff(on);
                        off.sendAtBeat(beatToSchedule+ 0.1);

                        // advance to next beat
                        beatToSchedule += 0.001;
                        beatToSchedule = Math.ceil(beatToSchedule * beat_division) /
                                                                beat_division;
                }
        }
}
// This code is based on the Logic script "Drum Probability Sequencer"
```

Appendix B

Addenda

B.1 System Identification

B.1.1 Music Notation

A table of music notes and rests is shown in Figure B.1

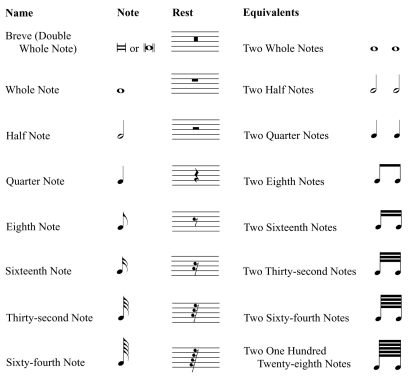


Figure B.1: Notes and Rests

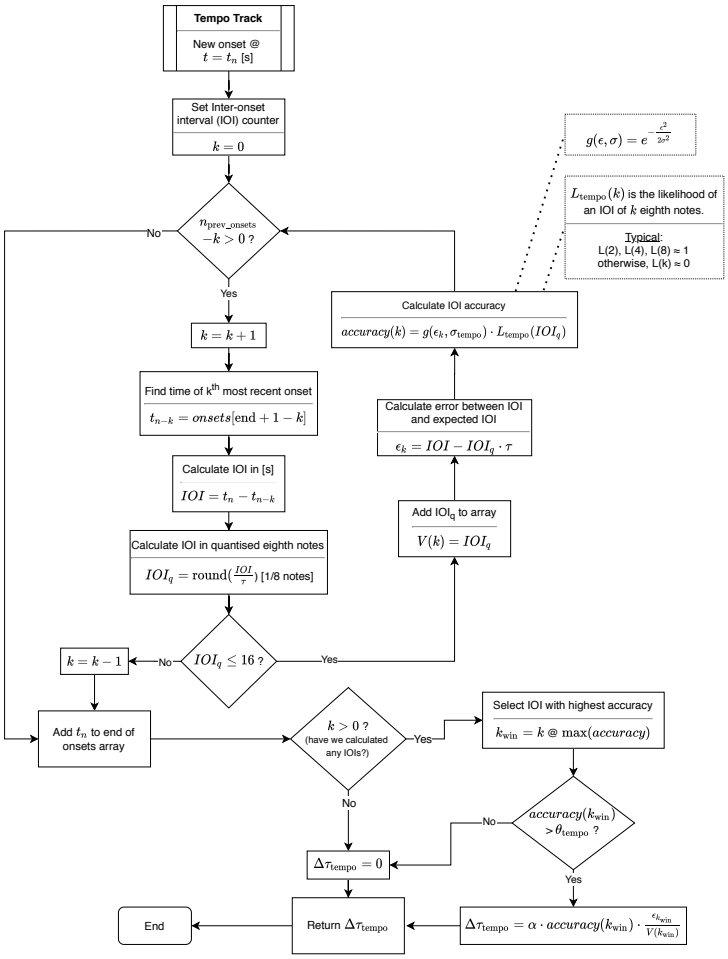


Figure B.2: B-Keeper: Tempo tracking process.

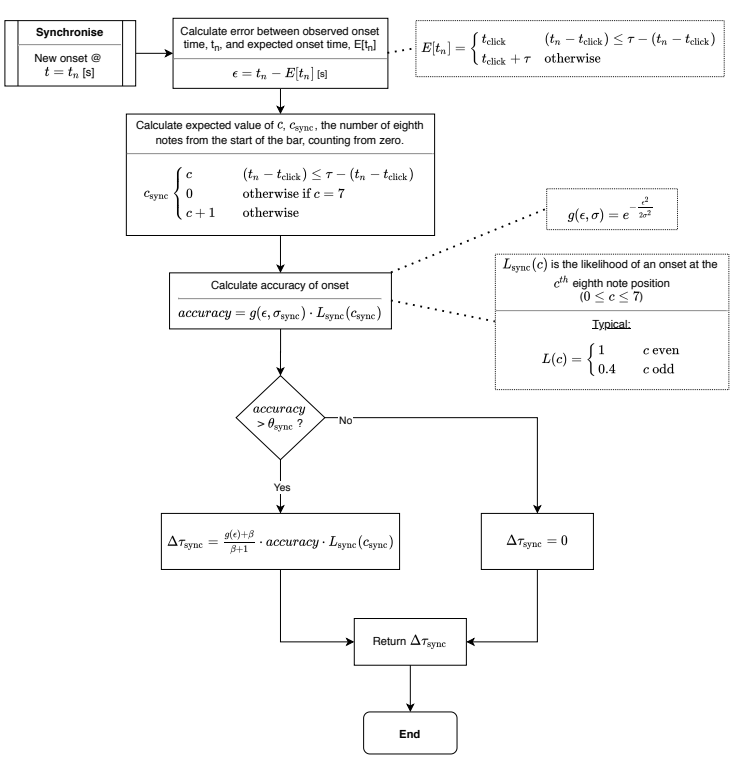


Figure B.3: B-Keeper: Synchronisation process.

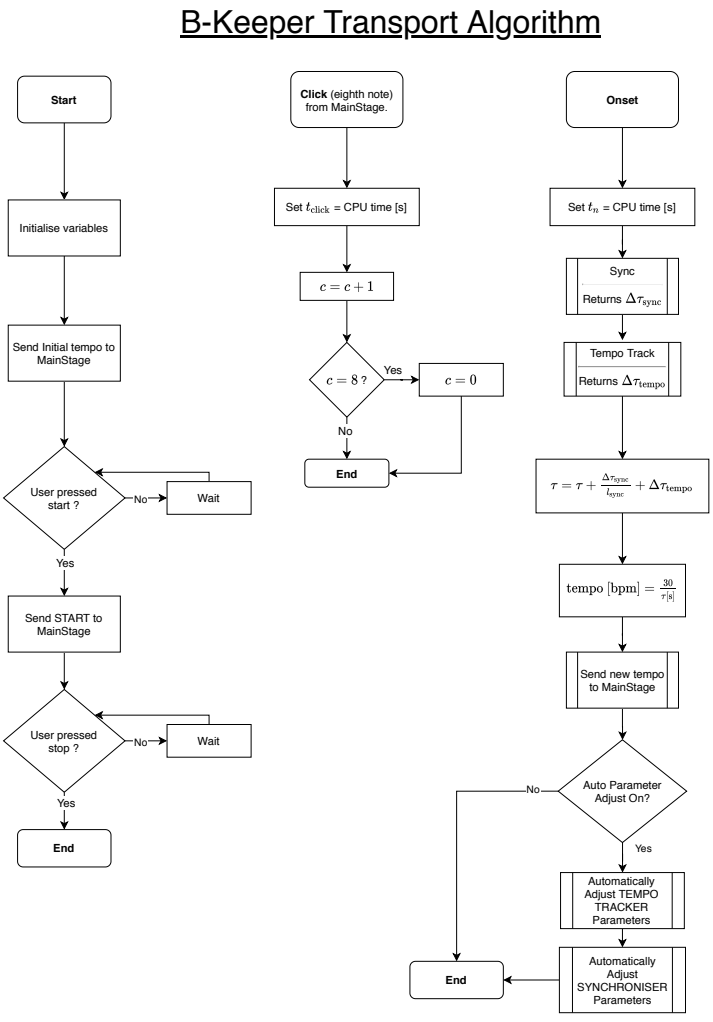


Figure B.4: Time keeping process developed for Python implementation of the B-Keeper algorithm. Time keeping was achieved using the `default_timer` module from the `timeit` package.

B.2 System Design

B.2.1 B-Keeper Algorithm

B.2.2 Approval of Ethics

The approval of Ethics is included here. It confirms that the ethics committee approved this research undertaking.

ETHICS APPLICATION FORM

Please Note:
Any person planning to undertake research in the Faculty of Engineering and the Built Environment (EBE) at the University of Cape Town is required to complete this form **before** collecting or analysing data. The objective of submitting this application *prior* to embarking on research is to ensure that the highest ethical standards in research, conducted under the auspices of the EBE Faculty, are met. Please ensure that you have read, and understood the **EBE Ethics in Research Handbook** (available from the UCT EBE, Research Ethics website) prior to completing this application form: http://www.ebe.uct.ac.za/ebe/research/ethics1

APPLICANT'S DETAILS		
Name of principal researcher, student or external applicant		
Department		
Preferred email address of applicant:		
If Student	Your Degree: e.g., MSc, PhD, etc.	
	Credit Value of Research: e.g., 60/120/180/360 etc.	
	Name of Supervisor (if supervised):	
If this is a researchcontract, indicate the source of funding/sponsorship		
Project Title		

I hereby undertake to carry out my research in such a way that:

- there is no apparent legal objection to the nature or the method of research; and
- the research will not compromise staff or students or the other responsibilities of the University;
- the stated objective will be achieved, and the findings will have a high degree of validity;
- limitations and alternative interpretations will be considered;
- the findings could be subject to peer review and publicly available; and
- I will comply with the conventions of copyright and avoid any practice that would constitute plagiarism.

APPLICATION BY	Full name	Signature	Date
Principal Researcher/ Student/External applicant			
SUPPORTED BY	Full name	Signature	Date
Supervisor (where applicable)			

APPROVED BY	Full name	Signature	Date
HOD (or delegated nominee) Final authority for all applicants who have answered NO to all questions in Section 1; and for all Undergraduate research (Including Honours).	A/Prof F Nicolls pp J Buxey		7.09.2020
Chair: Faculty EIR Committee For applicants other than undergraduate students who have answered YES to any of the questions in Section 1.			

Figure B.5: Ethics Approval Forms