Music Score Localisation 2.0

Carlos Soto i6216792
P. Koutsogeorgos i6328191
D. Wickrameratne i6176139
Sree Kotala i6206796
Olmo Denegri i6333396

Department of Advanced Computing Sciences
Maastricht University
Maastricht, The Netherlands

***Abstract*—The absence of an automatic page turner that can localise a musician performing to the appropriate section of a music score poses problems. Particularly for musicians with visual impairments who must zoom into the sheet music to distinguish notes, greatly increasing the frequency of page turns. The paper proposes a novel adaptation of the Shazam algorithm combined with Monte Carlo mobile robot-like localisation to overcome the challenges of music score localisation and account for repetition disambiguation. Initial experiments show promising results for the novel adaptation of Shazam, and lackluster performance in using Monte Carlo localisation in this setting.**

I. Introduction

Page turning is essential to every musician using sheet music during practice and performance as compositions rarely fit on a single sheet or tablet display. Though a trivial but occasional motion of the hand for most people, the same cannot be said for the visually impaired. To be able to distinguish the notes, visually impaired performers often use tablets for sheet music, and may need to zoom in to such a degree that only a very short portion of the music piece fits on the screen, thereby significantly increasing the frequency of required page-turns. This can become a serious problem, especially if the musician is trying to practice where employed page turners may not be available, or too expensive.

This is where automatic page turners come into play: software that is capable not only of showing the musical score, but also of keeping track of the position in the song the musician is currently performing and automatically turning pages as necessary. We refer to this as the *automatic score following* problem. This research problem was tackled recently by a group of DACS Master's students (Dick, Kane, Laguarta, Montesantos, & Naryzhnyaya, 2022), who used a symbolic approach to solve the music score localisation problem for *monophonic* melodies. Various algorithms have already been implemented in an attempt to solve the problem at hand (Dick et al., 2022).

Tonara (Kite-Powell, 2012), is a commercial software that follows a musician's performance by scrolling on the score of music and adapting to tempo changes. However, this approach can follow the musician from start to finish, but lacks the ability to re-calibrate once the performer stops during a practice session. Moreover, software like Enote (Roberts, 2020) claim a future add-on that will allow for automatic page-turning, while others, such as PlayAlong (*PlayAlong*, n.d.) and Tomplay (*Tomplay*, n.d.), simply follow and scan the score at a given speed.

The paper presents a novel algorithm that handles polyphonic melodies and follows the musician as they practice and perform; localising live music recordings from the device's microphone to the appropriate section of sheet music. This localisation problem can be interpreted as a way of measuring the similarity between an expected audio signal and one received as input from a microphone. The software solves said problem through an algorithm that is inspired by the popular app Shazam. The song being practiced is represented as constellation map of frequency peaks in time, which is then converted into an array of hashes representing note pairs being played closely in time. Disambiguation between repetitions (very similar if not identical sections of a song) is attempted with an approach similar to that of Monte Carlo Localisation for mobile robots (MCL). Multiple matching algorithms to localise a snippet onto the song were developed and compared to find the one most suitable to real-time use of the software, as for the purposes of real time application is it imperative that a song snippet x seconds long is localised in less than x seconds. Therefore, the ideal length of a song snippet required to achieve satisfactory localisation was investigated. In order to optimise the speed of localisation, this paper investigates two methods: parallelisation (I/O and CPU) and using heuristics of the location of a musician's played snippet on the music sheet. Finally the overall performance of the model is analysed and a prototypical GUI is developed.

Our novel adaptation of algorithm used in Shazam (Wang, 2003) is used to answer the following research questions.

1) **Automatic Score Following Strategy**
    a) How can we measure the similarity between an expected audio signal and the one actually received as input?
    b) How can we represent the expected sound wave

from a musical score and the one received as input for the purposes of song snippet localisation?
    c) How can we make our model robust enough in the face of unexpected changes to the piece being played such as pace variation, repetitions, and mistakes?
    d) What strategy performs the best?
    e) How long does a song snippet need to be for our model to able to localise it to a satisfactory degree?
2) **Limits of our Strategy**
    a) How does our strategy handle different instruments?
    b) Is there an lower limit to the snippet length we can use to localise with?
    c) How does the model tolerate noise and pitch shifting?
    d) Do harmonics pose a problem to our model?
3) **Performance Evaluation**
    a) How does our approach perform in polyphonic melodies?
    b) How does the performance of our model improve as we decrease our search space?
    c) What is the computational performance of our model?
    d) What is the computational complexity of our model?

II. Assumptions

The scope and focus of this paper meant that optimising the optical musician recognition techniques (OMR) presented in Dick et al. (2022) would not be an objective. The OMR techniques used by Dick et al. (2022) fell short of distinguishing polyphonic notes and other musical symbols, such as accidentals, clefs and bar-lines. To overcome the shortcomings of Dick et al. (2022)'s OMR techniques, this paper used Musical Instrument Digital Interface (MIDI) in order to create a perfect representation of all the notes in a music sheet along with various useful metadata. MIDI files are further explained in Section IV-A. Using MIDI files to create reference songs mean that **this paper assumes that: there already exists a perfect representation of all the notes** in a song. In addition, future sections in this paper will show that localising a snippet from an entire song proves incredibly difficult and computationally demanding. As a result, this paper reduces the search space by **assuming that the user is able indicate the location on a sheet of music, from which they would like to play.**

III. Pre-Requisites

A. Shazam

A flexible search algorithm was presented to identify a snippet of a song from a database of millions of commercially available songs - used by the company Shazam (Wang, 2003). This algorithm will henceforth be referred to as the Shazam algorithm. The Shazam algorithm plots the peak frequencies of a song's spectrogram (Fig. 1A) onto a "constellation map" by recording the brightest peak frequencies for every STFT window, and concatenating them into a two dimensional time frequency plot (Fig. 1B). The constellation map is what is referred to as an *acoustic fingerprint*: a unique representation of the peak frequencies at each time instance throughout a song. Each point on a constellation map acts as an anchor point to a target zone containing peaks ahead in time on the map (Fig. 1C). Diagonals are drawn from the anchor point to each point in its respective target zone. These diagonals are used to create a combinatorial hash, encoding the frequency and time difference between peaks (Fig. 1D). This process is repeated for an entire database of songs to create a unique fingerprint for each song. When a candidate song needs to be identified from this database, a 15 second clip is recorded, fingerprinted and then compared to the entire database of fingerprints. If there appears to be a match between the fingerprints and its encoded information, a title is returned to the user.

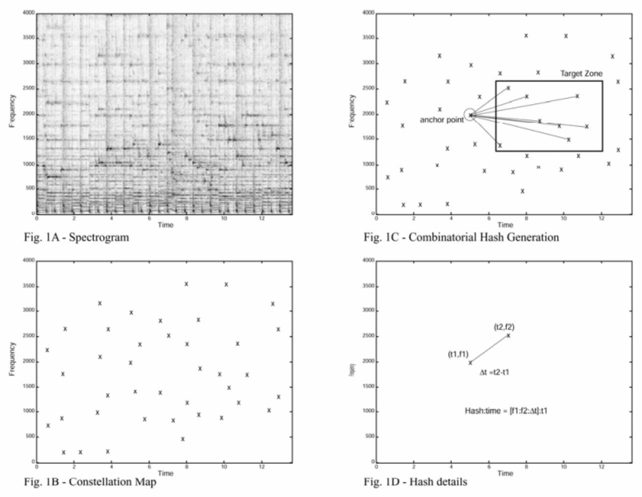

Fig. 1. A representation of an acoustic fingerprint (Wang, 2003).

Over the past few decades, the Shazam algorithm has proven to be robust against noise and distortion in the signal song, while being computationally effective and scalable. The most noteworthy aspect of this algorithm is its localising speed when identifying a song from a database of a million songs (Wang, 2003). These reasons justify the use of the Shazam algorithm as the foundation of our localisation problem. The Shazam algorithm on its own, is unable to solve the automatic page turner problem due to its inability to localise exact positions in songs and account for repeating sections. To circumvent this issue, we adapt the Shazam algorithm to use the following inputs:

Algorithm	**Shazam**	**Adapted Shazam**
Database	Millions of Songs	Sub-sets of Reference Song
Search Criteria	15s Recording	3s Recording
Search Result	Song Name and Artist	Point(s) in Time of Song

These adaptations allow for the localisation of short recorded snippets to specific sections within the songs. Although this isn't a complete solution to repetition disambiguation. The adapted Shazam is the foundation of the algorithms discussed in section IV-D

B. PANAKO

Six and Leman (2014) presents another acoustic fingerprinting system that is an extension of Shazam with the aim to make an algorithm which is more robust against time-scale and pitch modifications.

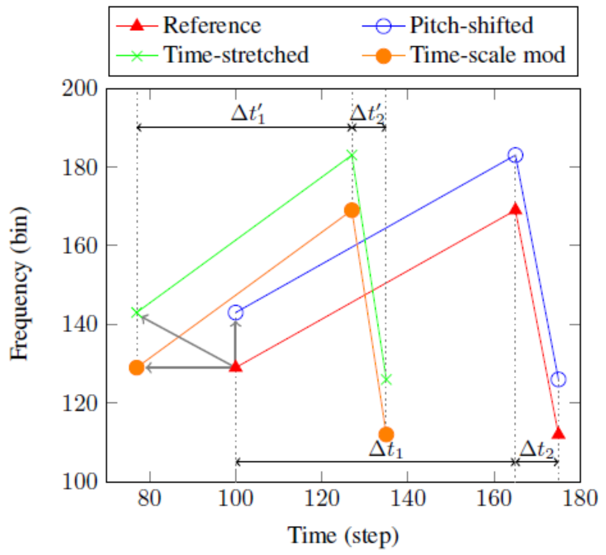

Fig. 2. The effect of time-scale and pitch modifications on a fingerprint. (Six & Leman, 2014).

In particular, the PANAKO approach differs to the Shazam approach in two main ways.

- PANAKO uses Q-transforms instead of spectrograms: varying the the window dimension of the time-frequency representation based on the particular frequencies under analysis.
- PANAKO considers triplets of points which define two consecutive diagonals instead of just one. The idea behind this modification is that given a triplet $(t_0, f_0), (t_1, f_1), (t_2, f_2)$ which is converted into a triplet $(t'_0, f'_0), (t'_1, f'_1), (t'_2, f'_2)$ after a constant time dilation and constant pitch shifting, then the following hold:

$$\frac{t_2 - t_1}{t_3 - t_1} = \frac{t'_2 - t'_1}{t'_3 - t'_1},$$

and

$$f_1 - f_2 = f'_1 - f'_2.$$

Therefore by encoding these constants into the hash created for the triplet, the algorithm has a better chance of matching fingerprints which have undergone time dilation or pitch shifting.

IV. Methods

A. Creating a reference song

In order to create reference notes of a song, MIDI files were used. MIDI files contain a digital representation of all notes played in a song by a particular instrument, the duration for which they were held and when they were played since the start of the song; thus making MIDI files a viable candidate to create a reference song for localisation. The MIDI files were converted to audio format (.wav) via a python library named FluidSynth (Hanappe, Nentwig, Schmitt, Green, & Letz, 2022). FluidSynth allowed for different instrument sounds to be applied onto the MIDI notes, which meant that the same song could be tested with different instruments. In addition, functions from the FluidSynth library could be embedded in a script rather than requiring the musician to pre-own a Digital Audio Workspace (DAW) - which also converts MIDI to audio, but requires more space on the user's operating system.

B. Audio Processing

Each time a reference song or recorded snippet are handled, they are passed through a signal processing pipeline (sp-pipeline). A visualisation of the pipeline is seen in Fig. 3.

1) Denoising: Raw audio files, reference songs or recorded snippets are first passed through a 2-level Wavelet denoising filter to account for irregularities in recordings between individuals such as microphone positions, background noise and recording environments.

2) Spectrogram building: These denoised signals are then passed through a spectrogram builder that uses Short-Term Fourier Transform (STFT) to generate a spectrogram using non overlapping Hann windows of 22ms in length. Non-overlapping segments were chosen to improve the distinction of peaks between neighbouring time intervals.

C. Constellation map

The final stage of the sp-pipeline includes creating constellation maps from the spectrograms. At each time interval in the spectrogram, the frequency of the highest peaks are marked, a minimum peak distance of ±100 Hz is included to ensure an even spread of points across the spectrum. These time-frequency marks are then collected to generate a map of points corresponding to the interesting features of the recording. This process is the basis for the Shazam algorithm.

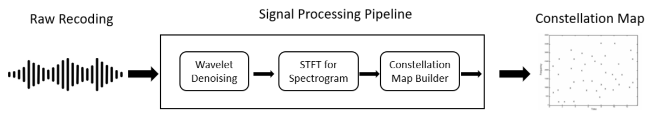

Fig. 3. Visualisation of the signal processing pipeline

D. Matching algorithms

We now describe and briefly compare the algorithms we developed in order to perform the localisation task given the constellation maps of the reference song and the recorded snippet.

1) Hashing: Our first attempt involves comparison of hashes in a similar manner as performed in the Shazam algorithm. To start with, given a constellation map, the hashes for the diagonals are computed as follows. For a diagonal consisting of the points (t_0, f_0), (t_1, f_1) (where $t_0 \leq t_1$), let $d = t_1 - t_0$. Now given a maximal frequency F which we assume that none of our constellation map frequencies surpass (23.000 Hz in our case), we calculate

$$f_0' = \text{int}\left(\frac{f_0}{F} \cdot 2^{10}\right),$$
$$f_1' = \text{int}\left(\frac{f_1}{F} \cdot 2^{10}\right).$$

Thus, f_0' and f_1' are 10-bit integer representations of the two frequencies which create the diagonal. The following integer is then stored as the final hash representing the diagonal:

$$\text{int(d)} \cdot 2^{20} + f_1' \cdot 2^{10} + f_0'.$$

We say that a hash is encountered at time t if it is generated from a diagonal of the form (t_0, f_0), (t_1, f_1) in the constellation map.

Given the generated hashes a dictionary is created, in which each key is a hash and its corresponding entry is a list of the time instances at which this hash is encountered. Note that due to repetitions within the song, it is possible that this list contains more than one element. To perform the matching, the hash dictionaries for both the reference song, and the recorded snippet constellation maps are created. The algorithm then searches whether each hash found in the snippet dictionary is also contained in the reference song dictionary. If so, a match is recorded for each time instance found in the list which is the entry of said hash in the reference song dictionary. Once the above process is finished for all the hashes in the snippet dictionary, the algorithm returns a list containing the time instances at which most matches were found.

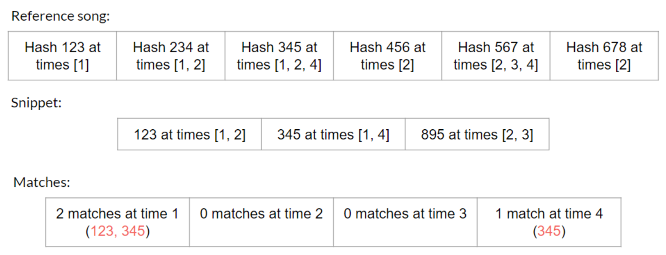

Fig. 4. A visualisation of the hashing method.

2) Sliding window (first version): A second method was developed to perform matching while taking into account the structure of the constellation maps with respect to time. In particular, once the hashes have been created from the diagonals, they are stored in an array in order of creation, which is increasing with respect to time. Note that multiple consecutive hashes in the array might be encountered at the same time instance in the constellation map. For this reason, a dictionary associating each index of the array to the time at which the corresponding hash is encountered is also generated. Once again we generate the above objects for both constellation maps. The algorithm then "slides" the recorded snippet array over the reference song array and produces a list containing the indices i of the reference song array such that whenever the 0'th index of the snippet array is aligned with i, a maximum number of matches is found. The times corresponding to said indices are then retrieved and returned by the matching algorithm.

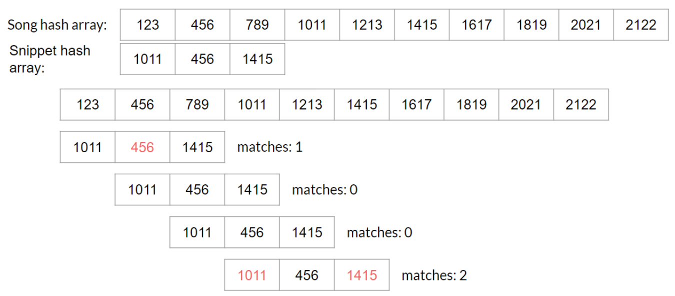

Fig. 5. A visualisation of the sliding window v1 method.

3) Sliding window (second version): The second sliding window method differs from the first one as follows: instead of storing the hashes in an array, the algorithm produces a dictionary the keys of which are the time instances found in the constellation map, and the corresponding entries are sets of hashes encountered at that time instance. The entries are then converted into a list, and the algorithm proceeds by "sliding" the list of the hash sets of the recorded snippet over the hash set list of the reference song. For each time instance the size of the intersection of the two sets is calculated, and those sizes are added to produce the matching score for that particular time instance. The algorithm as usual returns a list of the time instances with maximal matching score.

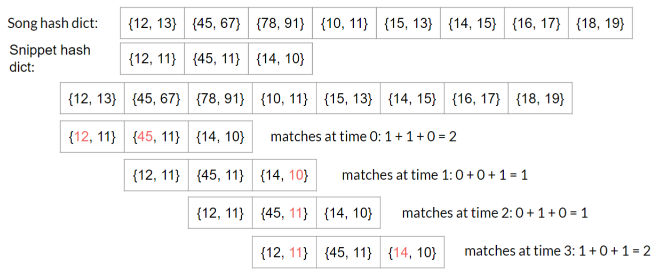
Song hash dict:
{12, 13} {45, 67} {78, 91} {10, 11} {15, 13} {14, 15} {16, 17} {18, 19}
Snippet hash dict:
{12, 11} {45, 11} {14, 10}
{12, 13} {45, 67} {78, 91} {10, 11} {15, 13} {14, 15} {16, 17} {18, 19}
{12, 11} {45, 11} {14, 10} matches at time 0: 1 + 1 + 0 = 2
{12, 11} {45, 11} {14, 10} matches at time 1: 0 + 0 + 1 = 1
{12, 11} {45, 11} {14, 10} matches at time 2: 0 + 1 + 0 = 1
{12, 11} {45, 11} {14, 10} matches at time 3: 1 + 0 + 1 = 2

Fig. 6. A visualisation of the sliding window v2 method.

4) PANAKO: Both the regular hashing method as well as the first sliding window method were also tested for the PANAKO extension of the Shazam algorithm. Two algorithms were developed based on those described above, with the only difference being in the hash generation. In particular, for the PANAKO extension hashes are not generated from diagonals in the constellation map, but rather from triplets as explained in III-B.

5) Direct diagonal comparison: One final method that was developed based on the first version of the sliding window approach is the one that follows. Instead of storing the hashes generated by the diagonals in an $n \times 1$ array (where n is the total number of diagonals considered), an $n \times 3$ array is created, the rows of which consist of the two frequencies and the time difference which define each diagonal. The sliding window algorithm then proceeds as explained above, only that in this case a match is found whenever the values of each row in the snippet array are within a certain interval centered around the corresponding values of the reference song array.

This algorithm proved to be significantly slower compared to the rest in testing and was therefore not tested or developed further.

E. Monte Carlo localisation

So far, we have considered different matching algorithms, where an independent song snippet is localised. However, no past observations are taken into account. These matching algorithms naively assume that the localisation of a song snippet does not depend on what was previously observed. This is clearly not the case. Taking previous observations into consideration could be beneficial, as it would, in principle, allow us to be more precise when localising, take errors into account and disambiguate repetitions.

The *song snippet localisation* task is in part analogous to the *mobile robot localisation* task, where a robot tries to find its position in a known map by using the subsequent observations taken by the robot. A powerful technique used to tackle robot localisation is *Monte Carlo localisation* (Dellaert, Fox, Burgard, & Thrun, 1999). We hypothesize that this technique when combined with any of the matching algorithms presented before, such as PANAKO, will make the strategy more robust and will be able to disambiguate repetitions and account for errors.

The main idea of such algorithm is to represent the probability density involved by maintaining a set of samples that are randomly drawn from it. By using a sampling-based representation a localisation method that can represent arbitrary distributions can be obtained. In robot localisation, we are interested in estimating the state of the robot x_k at the current time-step k, given knowledge about the initial state and all measurements $Z^k = \{z_k, i = 1...k\}$ up to the current time. More specifically, this method contains one initial phase and two phases that iteratively repeat:

1) Initialisation phase: In sampling-based methods one represents the density $p(x_k|Z^k)$ by a set of N random samples or particles, $S_k = s_k^i; i = 1..N$ drawn from it. Hence, in our initialisation phase we create an initial set of particles S_1, where each particle represents a time point in the reference song. The initial particles are evenly spaced across the reference song. We experiment with different number of particles in our study.

2) Prediction phase: In this phase we start from the set of particles S_{k-1} computed in the previous iteration, and:

- for each particle s_{k-1}^i:
  Draw one sample $s'i_k$ from $p(x_k|s_{k-1}^i, u_{k-1})$
  In doing so a new set S'_k is obtained that approximates a random sample from the predictive density $p(x_k|Z^{k-1})$
  The prime in S'_k indicates that we have not yet incorporated any sensor measurement at time k.

  For each particle i, we calculate the probability $p(x_k|s_{k-1}^i, u_{k-1})$ by:
  1) Adding the time elapsed since the last iteration to the particle.
  2) Assigning a maximum score to the point in time where the particle is, and a score that decreases linearly as we move forward or backwards. The total length of the interval centered at the time of the particle is 2 seconds
  3) Give a small score to points outside of the interval, to account for errors.
  4) Normalise all scores to obtain a valid probability distribution.

3) Update phase: In the second phase we take into account the measurement z_k, and weight each of the samples in S'_k, and the weight each of the samples in S'_k by the weight $m_k^i = p(z_k|s'^i_k)$, i.e., the likelihood of s'^i_k given z_k. We then obtain S_k by resampling from this weighted set:

- for j=1..N:
  draw one S_k sample s_k^j from $\{s'^i_k, m_k^i\}$

A new set S_k is obtained that approximates a random sample from $p(x_k|Z^k)$

The probability $p(z_k|s'^i_k)$ is calculated by using the scores from the desired matching algorithm. Our matching algorithms give a score to each time point in the song, which is proportional to the probability of the observation. In an attempt to reduce the effect of incorrect measurements/labelling, when the score for a time t is not 0, every time point in an interval around t will also get the same score at t. Right now this interval is 0.5 seconds. Every other point gets a small score to account for errors. For instance, if time 5.015 seconds gets a score of 4, then all timepoints contained in the interval $(5.015 - n, 5.015 + n)$ seconds will get a score of 4.

F. GUI

A GUI was devised as part of an application in order to visualise the localisation and test the algorithm on an iOS device. The software was programmed with the Kivy python library. The application required MIDI files to be pre-loaded before being uploaded onto the iOS device. Once the application was opened, the home screen displayed the pre-loaded MIDI files. Upon choosing a MIDI file, FluidSynth would convert the MIDI file to audio and store it locally, and then extract its MIDI notes and display them as shown in Fig. 7. As the user plays their instrument the localised notes would be highlighted in green, and the screen scrolls along the x-axis in order to keep up with the user. Due to the limited time available, it was unfortunately not integrated with any of the algorithms.

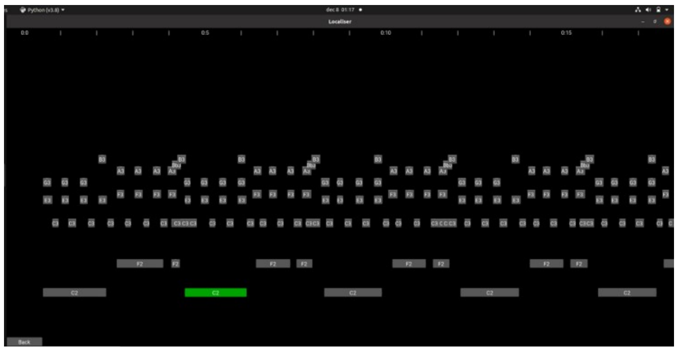

Fig. 7. A depiction of the developed GUI

G. Early attempts at Parallelisation

Seeing how one of the major obstacles to a successful practical application of this prototype software is the low speed of localisation in real-time, speeding up localisation is of utmost importance. For this purpose parallelisation of the code that handles localisation was attempted.

The idea is to split the reference song into n segments and then localise the snippets from the microphone in parallel onto each of the segments in a different process: as the performance of localisation is greatly affected by the size of the search space, localising a snippet onto a segment should be faster than localising it onto the whole reference constellation map.

Given an integer number n of desired segments and the length l of the hash array of song snippets to be localised, the reference song hash array was split into n overlapping segments, with the overlap having length $l/2$, to make sure that snippets falling at the intersection of segments could be successfully localised. Afterwards both multithreading and multiprocessing were implemented through the standard python library *concurrent.futures* library: in the former case every segment was assigned to a different thread and all the n localisation processes were run in parallel, in the latter case the *concurrent.futures* library independently handled assigning the segments with their respective localisation process to the different processors. Values for n used in these experiments were only 5 and 10. Unfortunately both approaches significantly slowed localisation down: multithreading caused a roughly 7-times factor slow-down, while multiprocessing performed better but still caused a slow-down by a factor of roughly 3. Multiprocessing performing better than multithreading is indicative of localisation being a CPU-bound computer task, and not an I/O-bound computer task. The tests were run on a 2015 MacBook Air having fewer cores than the 2020 iPad Pro, hence further investigation with newer hardware is recommended.

The code implementation for all described methods can be found here: https://github.com/csotogd/Music-Score-Localization-2.0

V. Experiments

In order to evaluate the developed strategies, three songs were used. Each song had one or more recordings made by an amateur pianist and bassist. For brevity, the songs used will be referred to by their abbreviated name and the different recorded takes will be denoted by a superscript indicating the number of the take.

- **Claire de Lune, by Claude Debussy (CDL):** A non-repetitive song filled with polyphonic notes, originally played on piano. CDL had one take (CDL^1) recorded on piano and contained a few correct notes. Due to its musical complexity, there were numerous mistakes, skipped notes and notes played at a slower tempo.
- **Prelude in C Major, by Bach (PCM):** A song composed of monophonic notes that repeat within an 8-beat interval, originally played on piano. PCM had three takes recorded. PCM^1 contained audible background noise and a few wrong notes throughout the recording. PCM^2 and PCM^3 had softer background noise but managed to contain fewer errors compared to PCM^1. All three recordings had sections where the pianist struggled to keep up with the original tempo.
- **The Ballad of John and Yoko, by The Beatles ($TBJY$):** A highly repetitive song composed of monophonic notes, originally played on bass. Three recordings were made for $TBJY$. $TBJY^1$ was played on bass and contained minimal errors and kept a good tempo. However, the clarity of each note was often drowned by the vibration of the heavy bass strings and the sound of fingers sliding across the fret board. $TBJY^2$ and $TBJY^3$ were both played on piano. $TBJY^2$ and $TBJY^3$ had wrong notes being played or notes being skipped, every few seconds. All three takes had minimal background noise.

Each of the recordings were manually labelled. For each song a set of labels $\{(t^1_{ref}, t^1_{sni}), ..., (t^n_{ref}, t^n_{sni})\}$, where t_{ref} refers to a time in the reference song (ground truth we want to predict), and t_{sni} refers to the time in the recorded song in which the sound of t_{ref} appears. This labelling is very time consuming, which is what prevented us from evaluating in a larger number of songs.

In order to evaluate our matching algorithms we do the following:
For each tuple (t^j_{ref}, t^j_{sni}):

1) Get snippet of the recorded song of a certain length, starting at t^j_{sni}
2) localise the recorded snippet in the reference song and obtain a time
3) Assign a score as can be observed in Fig. 8:

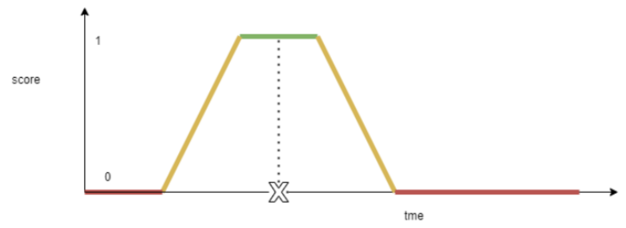

Fig. 8. Score evaluation function visualised, X indicates the true label t^j_{ref}.

- The predicted point gets a score of 1 if it is contained in an interval of 2 seconds, whose center is the true label t^j_{ref}
- The prediction gets a linearly decaying score if it lies in an interval of length 2 seconds which happens right after/before the central interval.
- The score is 0 if the predicted point is 3 seconds or further away from the true label t^j_{ref}.

The scores obtained for all tuples are averaged to obtain a final score. Each devised algorithm was used to localise a recorded take to itself and then to its reference song. By localising a recording to itself first, we establish a control that gives us an idea of which algorithm's scores are the most valid. Ideally, localising a recorded take to itself should yield an average score of 1. Thus, ensuring that localising a recorded take to its reference song yields the most valid results.

VI. Results and Discussion

A. Experimental Results

To be concise, the most important and note-worthy results will be displayed in this section. For the complete set of results, please refer the Appendix (Section A and B). Each answered or attempted research question will be in bold text and referenced. **(RQ 3b) All the results shown are from the reduced search space method which is four times faster than the regular search space method.** The results are divided into two parts: The best matching algorithm and MonteCarlo robot localisation.

TABLE I
The obtained average scores with hashing used by the Shazam algorithm for snippets of 3s, 5s and 10s from each recorded take. Each recorded take is **localised to itself.**

Song	Score (3s)	Match time (3s)	Score (5s)	Match time (5s)	Score (10s)	Match time (10s)
CDL^1	1	1.61	1	1.74	1	2.29
PCM^1	1	1.17	1	1.36	1	1.78
PCM^2	1	1.17	1	1.39	1	1.84
PCM^3	1	1.28	1	1.4	1	1.89
$TBJY^1$	1	2.43	1	3.14	1	4.83
$TBJY^2$	1	2.28	1	2.88	1	4.55
$TBJY^3$	1	2.36	1	2.99	1	4.59

1) Matching algorithms without MC: Table I shows that the standard hashing method used in the Shazam algorithm was able to perfectly localise each snippet of varied lengths in a recorded take within its respective recorded take. Table II shows the localisation scores and matching times for each song of various snippet length when localised to its reference song while using the hashing method by Shazam. When comparing the recorded take to itself and its respective reference song, the hashing used by the Shazam algorithm yielded the best results compared to every other investigated algorithm in the shortest amount of time.

TABLE II
The obtained average scores with hashing used by the Shazam algorithm for snippets of 3s, 5s and 10s from each recorded take. Each recorded take is **localised to its respective reference song.**

Song	Score (3s)	Match time (3s)	Score (5s)	Match time (5s)	Score (10s)	Match time (10s)
CDL^1	0.69	1.6	0.69	1.73	0.77	2.19
PCM^1	0.85	1.12	0.88	1.27	0.96	1.67
PCM^2	0.92	1.14	0.96	1.27	1	1.67
PCM^3	0.93	1.18	0.97	1.36	1	1.81
$TBJY^1$	0.7	3.26	0.8	4.52	0.9	7.7
$TBJY^2$	0.93	3.16	0.94	4.41	0.97	7.57
$TBJY^3$	0.94	3.28	0.95	4.57	0.97	7.78

(RQ 1d) The standard hashing by Shazam is faster due to two reasons: its relatively simple hash generation and constant matching complexity. Each hash only requires three components, whereas PANAKO requires seven. In terms of complexity, hash creation in Shazam is $O(10m)$ where m is the number of peak-frequency-time measurements from the constellation map. For each point on the constellation map, 10 points in the future are searched in order to create hashes. PANAKO searches a further 10 points for each one of the initially searched points. Therefore, PANAKO has a complexity of $O(100m)$ and is too slow to be used for localisation (Appendix A, Tables III, V, VII). In addition, the standard hashing algorithm by Shazam stores hashes in a dictionary and they can be localised with a simple inference, rather than having to iterate through an array of hashes as is done with the sliding window approach. Therefore, the matching time with hashes in the Shazam algorithm is independant of the number of hashes. **(RQ 1d) Moreover, hashing by Shazam results in perfect scores for all recorded takes when compared to themselves.** This is a good indication that the algorithm can be trusted to give valid results when localising a recorded take to its respective reference song. As a result, the hashing method by Shazam was deemed the most suitable algorithm to localise recorded takes to their respective reference song. Henceforth, any reference to our model in this section refers to one that uses the hashing method by Shazam.

(RQ 2c) The performance of our model is sensitive to the presence of audible background noise or noise made when playing the instruments themselves. Both PCM^1 and $TBJY^1$ were scored less than the other takes of their respective songs. We argue that the presence of audible background noise causes unwanted peaks on the constellation map, thus hindering the value of information stored in the hashes. It may be worth investigating a more rigorous signal processing method in order to eliminate sharp background noises such as sounds made by playing the instruments resulting in non-white noise that the current sp-pipeline is unable to handle. Another suggestion would be to look further ahead in time when creating hashes so that they contain the closest peak-frequencies caused by the instrument, rather than the closest peak-frequencies caused by other background noise.

(RQ 1c) It is unclear if our model is robust against tempo variation in the musician's performance. The musician had trouble with keeping tempo in PCM^1, PCM^2 and PCM^3 - but said recordings still scored better than $TBJY^1$, which was relatively on tempo. The difference between these recorded takes were the levels of background noise. Further testing is required to see how robust our model is against tempo variation; this would be preferably done by recording songs with the same level of background noise and varying degrees of tempo variation.

(RQ 3a) Table II shows that our model performs better on monophonic melodies compared to polyphonic melodies. We suspect that this difference is caused by the Hann window of 22ms used in the creation of the constellation map. The individual notes that make up a polyphonic melody or chord may be struck at intervals greater than 22ms, or lie within the boundary between two 22ms windows, thus resulting in being assigned to a different time point on the constellation map. Furthermore, in order to create sparsity in the constellation map, the peak frequencies within a $\mp$ 100Hz range from the highest peak-frequencies are eliminated. As a result, important frequencies may be lost. A longer Hann window and a shorter frequency cut-off range may result in the model performing better with polyphonic notes at the cost of a loss in time resolution. **(RQ 3a) Nonetheless, it is important to note that the model can handle polyphonic melodies.**

(RQ 1e) The optimal snippet length was found to be 5 seconds. Table II shows that the best scores were obtained when localising snippets of 10 seconds. However, updating a musician's position on a music score after they have played for 10 seconds seemed impractical. An amateur musician is bound to require a more frequent indication of their position in order to assist their playing. In addition, as a musician reaches the end of a page, playing the notes within a 10 second interval may be difficult if some notes lie on the next page and the musician is not familiar with the piece they are playing. A snippet of 3 seconds yielded satisfactory results. However the time our model took to localise the snippets were greater than 3 seconds for the recorded takes with high repetitions. A snippet of 5 seconds yielded strong results with an appropriate matching time, thus allowing us to believe that our model works best when localising snippets of 5 seconds. Experiments with snippet lengths shorter than 3 seconds were also run (0.5, 1 and 2 seconds), but such lengths were quickly discarded as not only were they yielding comparable or lower scores to 3-seconds long snippets, all matching algorithms were also sometimes completely unable to localise such short snippets. **(RQ 2b) From these experiments we conclude that 3 seconds is the minimum length necessary for successful snippet localization**.

(RQ 1a 1b 2a) Lastly, it can be seen that our model was able to handle different instruments to that of the originally used instrument for a song. $TBJY$ is originally played on bass, but the model was more successful in localising the same notes played on a piano. **(RQ 1a 1b 2d) Moreover, the sp-pipeline disregards harmonics or overtones, resulting in the localisation algorithm producing very similar constellation maps at the end of the sp-pipeline, irrespective of the instrument being played**. This is done as harmonics and overtones are not integral to the localisation of snippets using our adapted method.

2) MonteCarlo Robot-Localisation results: The proposed Montecarlo localisation strategy was unable to localise the song snippet for more than 3 consecutive snippets, across all songs and versions. We hypothesize this is due to different factors:

- In the original robot-localisation, when the distance from the robot to the wall is measured, the obtained measure contains no errors or are not significant. **(RQ 1c) In the music case, the matching algorithms (such as PANAKO or Shazam) which are intrinsically robust to noise and pitch variation, sometimes fail to localise the snippet by more than 10 seconds**. This completely spoils the calculation of the probability $p(z_k|s_k'^i)$. Giving a small probability to time points that receive no score by the matching algorithm, was tried but yielded no satisfactory results.

- In the original robot-localisation, when the robot moves n meters, we can be sure that it moves n meters. In the music localisation case, it may be the case that between a recorded song snippet and the next one, n seconds elapse, yet according to the music sheet $2n$ seconds should have elapsed. This is a very common situation, as musicians usually play slower and faster and skip silences. This results in the calculation of $p(x_k|s_{k-1}^i, u_{k-1})$, where we are adding to each particle the time between the previous recording and the next one. Allowing the particle to be in a certain interval instead of a single point did also not help.

B. Complexity Analysis

(RQ 3c 3d) The total complexity, C_{tot}, is comprised of five processes that vary in their levels of complexity, namely: signal processing, hash creation, MonteCarlo (MC) particle creation, search-space reduction and the localising algorithm.

- The complexity of signal processing, C_{SP}, is $O(n)$ where n is the number of frequency-time measurements within a signal. The STFT and Wavelet filters require iterating through the entire data set of a signal. Therefore signal processing is linearly complex.
- The complexity of hash creation, C_{HC}, is $O(10m)$ where m is the number of peak-frequency-time measurements from the constellation map.
- MC particle creation, C_{PC}, has a complexity of $O(p * log(p))$ where p is the number of particles that are created and sorted.
- Search-space reduction involves segmenting the amount of peak-frequency-time measurements from the constel-

lation map. Therefore, the complexity of search-space reduction, C_{SR} is $O(m)$ where m is the number of peak-frequency-time measurements from the constellation map.
- The complexity of the localising algorithm is itself comprised of two complexities, namely: the hash inference and MC particle matching. The complexity of the hash inference, C_{HI}, is $O(1)$ as it requires extracting the value of a key in a dictionary. MC matching has a complexity, C_{MC}, of $O(p^2)$ as every particle's position is updated with respect to every other particle - so the distribution of particles are iterated p times for p particles.

Therefore, C_{tot} in real-time can be given by the following expression:

$$C_{tot} = C^*_{SP} + C_{HC} + C^*_{HC} + C_{PC} + C_{SR} + C_{HI} + C_{MC} \quad (1)$$

where C^*_{HC} and C^*_{SP} are dynamically updated terms. C^*_{SP} is the complexity of the signal processing being performed as the musician inputs a signal. C^*_{SP} will depend on the number of notes played within a certain duration. C^*_{HC} is the complexity of the hash creation from the input signal. C^*_{HC} will also depend on the number of notes played within a certain duration. Therefore, C_{tot} can be written with big-O notation as:

$$\begin{aligned} C_{tot} = O(n) + O(10m) + O(10m^*) \\ + O(p * log(p)) + O(m) + O(1) + O(p^2) \end{aligned} \quad (2)$$

where n is the amount of frequency-time measurements from the musician's input signal, m is the number of peak-frequency-time measurements from the reference song, m^* is the number of peak-frequency-time measurements from the input signal, h is the number of hashes created by the reference song and p is the number of particles created. Provided that the hashes created from the reference song can be saved locally and loaded when needed, C_{HC} can be ignored. In addition, due to the poor performance of MCL in our model, we decided to discard it. Therefore, C_{PC} and C_{MC} may be ignored. As a result, the C_{tot} can be reduced to:

$$C_{tot} = O(n) + O(10m^*) + O(m) + O(1) \quad (3)$$

VII. Conclusion

We firmly believe that the hashing method adapted from the Shazam algorithm performs the best and constitutes a good baseline for an automatic score following software capable of handling both monophonic and polyphonic melodies. However we are aware that further testing is required: the two main obstacles to testing being that the manual labelling of recordings is very time-consuming and the fact that such labels are far from being perfectly accurate. Furthermore our findings show that MCL is not suited to the task of repetition disambiguation. We are confident that our scoring interval of 6 seconds (Figure 8) allows our model to be accurate to the nearest stave on a music sheet. Lastly, more effort should be put into attempting to parallelise the code and measuring its performance on either an actual iPadPro 2020 device or any machine with at least 4 or 6 cores.

VIII. Acknowledgement

We would like to thank Dr. Kurt Driessens. Kurt, thank you for your supervision and support during this project, your machine learning expertise and in particular, the enthusiastic attitude in meetings and shared excitement for the project. It surely motivated the group every time we came to meet with you. We would also like to thank Dr. Pietro Bonizzi, thank you for answering our many, many questions during meetings and after lectures, as well as brainstorming with us to think of ideas to consider and incorporate in order to make this a well rounded project.

References

Dellaert, F., Fox, D., Burgard, W., & Thrun, S. (1999). Monte carlo localization for mobile robots. In *1999 IEEE international conference on robotics and automation, marriott hotel, renaissance center, detroit, michigan, usa, may 10-15, 1999, proceedings* (pp. 1322–1328). IEEE Robotics and Automation Society. Retrieved from https://doi.org/10.1109/ROBOT.1999.772544 doi: 10.1109/ROBOT.1999.772544

Dick, T., Kane, E., Laguarta, P., Montesantos, G., & Naryzhnyaya, M. (2022, 06). Automatic musical score page turner.

Hanappe, P., Nentwig, M., Schmitt, A., Green, J., & Letz, S. (2022). *Fluidsynth user manual.* Retrieved from https://github.com/FluidSynth/fluidsynth/wiki/UserManual

Kite-Powell, J. (2012). *Tonara: An intelligent music app.* Forbes.

Playalong. (n.d.). Sheet Music Direct. Retrieved from https://www.sheetmusicdirect.com/playalong

Roberts, M. S. (2020). *Um, this genius sheet music app will listen to your playing and turn pages for you.* Classic FM.

Six, J., & Leman, M. (2014). PANAKO - A scalable acoustic fingerprinting system handling time-scale and pitch modification. In H. Wang, Y. Yang, & J. H. Lee (Eds.), *Proceedings of the 15th international society for music information retrieval conference, ISMIR 2014, taipei, taiwan, october 27-31, 2014* (pp. 259–264). Retrieved from http://www.terasoft.com.tw/conf/ismir2014/proceedings/T048_122_Paper.pdf

Tomplay. (n.d.). Tombooks Sarl. Retrieved from https://tomplay.com/

Wang, A. (2003). An industrial strength audio search algorithm. In *ISMIR 2003, 4th international conference on music information retrieval, baltimore, maryland, usa, october 27-30, 2003, proceedings.*

APPENDIX

A. Results without MCL: Reference vs Reference

TABLE III
THE OBTAINED AVERAGE SCORES WITH HASHING USED BY THE PANAKO ALGORITHM FOR SNIPPETS OF 3S, 5S AND 10S FROM EACH RECORDED TAKE. EACH RECORDED TAKE IS **LOCALISED TO ITSELF.**

Song	Score (3s)	Match time (3s)	Score (5s)	Match time (5s)	Score (10s)	Match time (10s)
CDL^1	0.08	3.82	0.1	4.12	0.31	4.85
PCM^1	0.08	4.37	0.08	4.73	0.15	5.62
PCM^2	0.08	4.31	0.13	4.64	0.19	5.48
PCM^3	0.07	4.8	0.11	5.2	0.29	6.12
$TBJY^1$	0.04	18.98	0.05	20.83	0.1	25.08
$TBJY^2$	0.04	18.23	0.05	19.75	0.11	23.65
$TBJY^3$	0.04	18.68	0.05	20.33	0.1	24.6

TABLE IV
THE OBTAINED AVERAGE SCORES WITH HASHING USED BY THE SHAZAM ALGORITHM FOR SNIPPETS OF 3S, 5S AND 10S FROM EACH RECORDED TAKE. EACH RECORDED TAKE IS **LOCALISED TO ITSELF.**

Song	Score (3s)	Match time (3s)	Score (5s)	Match time (5s)	Score (10s)	Match time (10s)
CDL^1	1	1.61	1	1.74	1	2.29
PCM^1	1	1.17	1	1.36	1	1.78
PCM^2	1	1.17	1	1.39	1	1.84
PCM^3	1	1.28	1	1.4	1	1.89
$TBJY^1$	1	2.43	1	3.14	1	4.83
$TBJY^2$	1	2.28	1	2.88	1	4.55
$TBJY^3$	1	2.36	1	2.99	1	4.59

TABLE V
THE OBTAINED AVERAGE SCORES WITH SLIDING WINDOWS (V1) AND HASHING USED BY THE PANAKO ALGORITHM FOR SNIPPETS OF 3S, 5S AND 10S FROM EACH RECORDED TAKE. EACH RECORDED TAKE IS **LOCALISED TO ITSELF.**

Song	Score (3s)	Match time (3s)	Score (5s)	Match time (5s)	Score (10s)	Match time (10s)
CDL^1	0.08	11.31	0.08	13.26	0.08	16.78
PCM^1	0.04	18.82	0.07	23.77	0.04	30.41
PCM^2	0.04	17.92	0.07	22.46	0.04	28.86
PCM^3	0.03	19.6	0.06	24.65	0.04	31.39
$TBJY^1$	0.03	73.36	0.03	89.97	0.03	115.72
$TBJY^2$	0.03	70.6	0.03	88.98	0.03	111.91
$TBJY^3$	0.03	73.32	0.03	89.11	0.03	116.44

TABLE VI
THE OBTAINED AVERAGE SCORES WITH SLIDING WINDOWS (V1) AND HASHING USED BY THE SHAZAM ALGORITHM FOR SNIPPETS OF 3S, 5S AND 10S FROM EACH RECORDED TAKE. EACH RECORDED TAKE IS **LOCALISED TO ITSELF.**

Song	Score (3s)	Match time (3s)	Score (5s)	Match time (5s)	Score (10s)	Match time (10s)
CDL^1	1	1.98	1	2.12	1	2.53
PCM^1	1	1.71	1	1.88	1	2.27
PCM^2	1	1.79	1	1.88	1	2.26
PCM^3	0.93	1.81	0.93	2.01	1	2.47
$TBJY^1$	0.9	5	0.93	5.71	0.98	7.41
$TBJY^2$	0.9	5.01	0.94	5.5	0.98	6.92
$TBJY^3$	0.9	4.99	0.93	5.68	0.98	7.12

TABLE VII
THE OBTAINED AVERAGE SCORES WITH SLIDING WINDOWS (V2) AND HASHING USED BY THE PANAKO ALGORITHM FOR SNIPPETS OF 3S, 5S AND 10S FROM EACH RECORDED TAKE. EACH RECORDED TAKE IS **LOCALISED TO ITSELF.**

Song	Score (3s)	Match time (3s)	Score (5s)	Match time (5s)	Score (10s)	Match time (10s)
CDL^1	0.08	3.98	0.08	4.32	0.08	5.24
PCM^1	0.04	4.72	0.04	5.12	0.04	6.95
PCM^2	0.04	4.75	0.04	5.24	0.04	6.04
PCM^3	0.03	5.17	0.03	5.83	0.03	6.74
$TBJY^1$	0.03	20.05	0.03	21.9	0.03	26.39
$TBJY^2$	0.03	18.97	0.03	21.14	0.03	25.74
$TBJY^3$	0.03	19.88	0.03	21.68	0.03	26.19

TABLE VIII
THE OBTAINED AVERAGE SCORES WITH SLIDING WINDOWS (V2) AND HASHING USED BY THE SHAZAM ALGORITHM FOR SNIPPETS OF 3S, 5S AND 10S FROM EACH RECORDED TAKE. EACH RECORDED TAKE IS **LOCALISED TO ITSELF.**

Song	Score (3s)	Match time (3s)	Score (5s)	Match time (5s)	Score (10s)	Match time (10s)
CDL^1	1	1.71	1	1.91	1	2.39
PCM^1	1	1.33	1	1.58	1	2.1
PCM^2	1	1.34	1	1.56	1	2.08
PCM^3	0.93	1.4	0.93	1.66	1	2.25
$TBJY^1$	0.88	3.03	0.94	3.99	0.98	6.22
$TBJY^2$	0.89	2.91	0.94	3.86	0.98	6
$TBJY^3$	0.88	3.01	0.94	4.06	0.98	6.21

B. Results without MCL: Recording vs Reference

TABLE IX
THE OBTAINED AVERAGE SCORES WITH HASHING USED BY THE PANAKO ALGORITHM FOR SNIPPETS OF 3S, 5S AND 10S FROM EACH RECORDED TAKE. EACH RECORDED TAKE IS **LOCALISED TO ITS RESPECTIVE REFERENCE SONG.**

Song	Score (3s)	Match time (3s)	Score (5s)	Match time (5s)	Score (10s)	Match time (10s)
CDL^1	0.08	3.74	0.08	4.03	0.35	4.74
PCM^1	0.08	4.25	0.08	4.51	0.15	5.27
PCM^2	0.08	4.2	0.13	4.47	0.19	5.28
PCM^3	0.07	4.54	0.11	4.88	0.29	5.68
$TBJY^1$	0.04	113.04	0.05	23.33	0.1	30.71
$TBJY^2$	0.03	19.95	0.03	22.99	0.11	31.37
$TBJY^3$	0.03	21.08	0.03	24.63	0.1	32.25

TABLE X
THE OBTAINED AVERAGE SCORES WITH HASHING USED BY THE SHAZAM ALGORITHM FOR SNIPPETS OF 3S, 5S AND 10S FROM EACH RECORDED TAKE. EACH RECORDED TAKE IS **LOCALISED TO ITS RESPECTIVE REFERENCE SONG.**

Song	Score (3s)	Match time (3s)	Score (5s)	Match time (5s)	Score (10s)	Match time (10s)
CDL^1	0.69	1.6	0.69	1.73	0.77	2.19
PCM^1	0.85	1.12	0.88	1.27	0.96	1.67
PCM^2	0.92	1.14	0.96	1.27	1	1.67
PCM^3	0.93	1.18	0.97	1.36	1	1.81
$TBJY^1$	0.7	3.26	0.8	4.52	0.9	7.7
$TBJY^2$	0.93	3.16	0.94	4.41	0.97	7.57
$TBJY^3$	0.94	3.28	0.95	4.57	0.97	7.78

TABLE XI
THE OBTAINED AVERAGE SCORES WITH SLIDING WINDOWS (V1) AND HASHING USED BY THE PANAKO ALGORITHM FOR SNIPPETS OF 3S, 5S AND 10S FROM EACH RECORDED TAKE. EACH RECORDED TAKE IS **LOCALISED TO ITS RESPECTIVE REFERENCE SONG.**

Song	Score (3s)	Match time (3s)	Score (5s)	Match time (5s)	Score (10s)	Match time (10s)
CDL^1	0.08	10.26	0.08	12.44	0.08	15.92
PCM^1	0.05	16.89	0.06	20.48	0.05	27.28
PCM^2	0.04	15.92	0.06	20.2	0.05	27.46
PCM^3	0.03	17.33	0.07	20.74	0.07	28.73
$TBJY^1$	0.03	93.32	0.03	113.33	0.03	124.13
$TBJY^2$	0.03	95.57	0.03	113.57	0.03	110.87
$TBJY^3$	0.03	100.5	0.03	119.41	0.03	115.02

TABLE XII
THE OBTAINED AVERAGE SCORES WITH SLIDING WINDOWS (V1) AND HASHING USED BY THE SHAZAM ALGORITHM FOR SNIPPETS OF 3S, 5S AND 10S FROM EACH RECORDED TAKE. EACH RECORDED TAKE IS **LOCALISED TO ITS RESPECTIVE REFERENCE SONG.**

Song	Score (3s)	Match time (3s)	Score (5s)	Match time (5s)	Score (10s)	Match time (10s)
CDL^1	0.69	1.94	0.43	2.12	0.38	2.53
PCM^1	0.71	1.69	0.72	1.85	0.71	2.26
PCM^2	0.73	1.66	0.68	1.85	0.73	2.27
PCM^3	0.72	1.78	0.59	1.97	0.61	2.45
$TBJY^1$	0.3	5.98	0.25	7.25	0.2	9.82
$TBJY^2$	0.36	5.79	0.45	6.91	0.04	9.21
$TBJY^3$	0.4	6.05	0.46	7.22	0.02	9.56

TABLE XIII
THE OBTAINED AVERAGE SCORES WITH SLIDING WINDOWS (V2) AND HASHING USED BY THE PANAKO ALGORITHM FOR SNIPPETS OF 3S, 5S AND 10S FROM EACH RECORDED TAKE. EACH RECORDED TAKE IS **LOCALISED TO ITS RESPECTIVE REFERENCE SONG.**

Song	Score (3s)	Match time (3s)	Score (5s)	Match time (5s)	Score (10s)	Match time (10s)
CDL^1	0.08	4.05	0.08	4.33	0.08	5.06
PCM^1	0.04	4.6	0.04	4.9	0.04	5.76
PCM^2	0.04	4.47	0.04	4.91	0.04	6.09
PCM^3	0.04	7.99	0.04	6.14	0.04	6.46
$TBJY^1$	0.03	23.17	0.03	26.15	0.03	33.68
$TBJY^2$	0.03	22.43	0.03	25.38	0.03	34.79
$TBJY^3$	0.03	24.54	0.03	27.01	0.03	35.17

TABLE XIV
THE OBTAINED AVERAGE SCORES WITH SLIDING WINDOWS (V2) AND HASHING USED BY THE SHAZAM ALGORITHM FOR SNIPPETS OF 3S, 5S AND 10S FROM EACH RECORDED TAKE. EACH RECORDED TAKE IS **LOCALISED TO ITS RESPECTIVE REFERENCE SONG.**

Song	Score (3s)	Match time (3s)	Score (5s)	Match time (5s)	Score (10s)	Match time (10s)
CDL^1	0.77	1.74	0.58	1.94	0.38	2.44
PCM^1	0.72	1.32	0.62	1.55	0.82	2.1
PCM^2	0.81	1.34	0.69	1.55	0.88	2.11
PCM^3	0.71	1.35	0.73	1.65	0.6	2.24
$TBJY^1$	0.33	4.78	0.34	6.17	0.2	9.74
$TBJY^2$	0.4	4.33	0.53	6.04	0.07	9.18
$TBJY^3$	0.39	4.39	0.5	6.09	0.02	9.41