Polyphonic Pitch Detection Algorithm for Guitar Audio

The code implements an algorithm for polyphonic pitch detection on guitar audio data. The goal is to identify and track the individual notes or pitches being played on the guitar. The algorithm achieves this by applying various logical conditions to the audio data. Let's break down the key components and logic conditions step by step:

1. Data Loading and Preparation

- The algorithm begins by loading data from score files, including FFT data, filter data, monophonic detection data, and envelope data. These data sets provide information about the guitar audio signal.

2. Parameters and Thresholds

- Various parameters are defined, such as `Pshp`, `Pshs`, `Pthp`, and `Pths`, which determine the thresholds used in the detection conditions.

3. Iteration over Chunks and Notes

- The algorithm iterates over chunks of the audio data (represented by variable `i`) and individual notes (represented by variable `j`).

4. Conditions for Note Detection

The algorithm employs several conditions to detect notes within the audio data:

- **Condition 1** (`if` statement):
    - Based on detecting harmonics of a note.
    - Checks if specific frequency components (`vectnote[j+12]` and `vectnote[j+19]`) satisfy amplitude and harmonic-related conditions.
    - If the conditions are met, the note is detected and stored in `vectnote[j]`.
- **Condition 2** (`elif` statement):
    - Based on detecting the waveform pattern of a note.
    - Checks if the monophonic detection data (`vectmono[j]`) and amplitude conditions are satisfied.
    - If the conditions are met, the note is detected and stored in `vectnote[j]`.
- **Conditions 3 and 4** (`elif` statements):
    - Specifically for detecting higher-frequency notes.
    - Involve amplitude checks and conditions related to adjacent notes and their harmonics.
    -

If the conditions are met, the note is detected and stored in `vectnote[j]`.

- **Conditions 5 and 6** (`elif` statements):
  - Focus on detecting specific intervals between notes, such as fifths and thirds.
  - Involve amplitude checks and frequency ratio calculations.
  - If the conditions are met, the note is detected and stored in `vectnote[j]`.
- **Fallback Condition** (`else` statement):
  - If none of the above conditions are met, the algorithm attempts to recover notes using previously stored data.
  - If the stored data suggests the presence of a note, it is re-detected and stored in `vectnote[j]`.

5. Attack and Allowance

- The algorithm calculates the attack value based on envelope data.
- The attack value dynamically adjusts the `allowance` variable, determining when note detection is enabled or disabled.
- If the attack value surpasses a certain threshold, the allowance is adjusted to allow note detection.

6. Visualization

- The algorithm includes visualization using Matplotlib to plot detected notes, conditions, and other relevant information.

7. Resetting Variables

- After processing a chunk, the algorithm resets various variables to prepare for the next iteration.

In summary, the algorithm uses a combination of amplitude conditions, harmonic checks, waveform patterns, and frequency ratios to detect and track notes in the guitar audio signal. The conditions are designed to capture different aspects of the audio signal corresponding to different note types and playing techniques on the guitar. The dynamic adjustment of the `allowance` based on attack values ensures accurate note detection while accounting for variations in the audio signal.

TODO

1. **Condition on Thirds and Chunks 23**:
   - Review and analyze the existing condition for thirds in the chord detection algorithm.
   - Consider different chord configurations within the C-A-G-E-D system (e.g., A, E, D) and identify how the current condition covers them.
   - Explore potential modifications to the thirds condition to encompass a broader range of chord configurations.
   - Investigate the issue at chunk 23 where the note "Si3" (B3) is incorrectly recognized as the fifth of "Mi2" (E2) and the note "La2" (A2) is not detected. Identify the root cause of this problem.
2.

Dealing with Fifths and Leaving Thirds Aside:

- Temporarily set aside the work on improving thirds detection and focus on addressing the issues related to fifths in the algorithm.
- Review the existing conditions for detecting fifths and evaluate their effectiveness in different chord contexts.
- Identify specific cases where the detection of fifths is failing or producing incorrect results.
- Develop and test new conditions or modifications to existing conditions that can enhance the detection accuracy of fifths.

3. **Comparison with G(open)**:

   - Analyze the chord detection results for the "G(open)" chord and compare them with the observations from other chords.
   - Specifically, focus on the detection of the note "Re1" (D1) and its absence in the results.
   - Investigate the possibility of introducing an additional condition that considers the context of polyphony (indicated by the variable "existsum").
   - Experiment with different thresholds or criteria for detecting certain notes in polyphonic contexts, and assess the impact on chord recognition accuracy.

4. **Understanding a Quote from May 31, 2020**:

   - Retrieve and review the statement from May 31, 2020, to gain a deeper understanding of the issues related to the octaves of fifths and the FFT-based condition.
   - Revisit the FFT-based condition and assess its role in detecting fifths and other harmonic relationships.
   - Experiment with variations of the FFT-based condition or explore alternative methods that can improve the detection of fifths while still maintaining its usefulness for other aspects of pitch detection.
   - Determine whether adjustments to the FFT-based condition can address the errors observed in recognizing the G major chord.

5. **Overall Algorithm Optimization**:

   - Continuously monitor and evaluate the algorithm's performance and accuracy across different chords and musical contexts.
   - Collaborate with domain experts or colleagues to gather insights and feedback on potential improvements to the algorithm.
   - Document and track changes made to the algorithm, including modifications to conditions, thresholds, and detection strategies.
   - Run extensive testing and validation on various musical samples to ensure that the algorithm's accuracy and reliability are improved across the board.
   - Consider conducting comparative analyses between the original and optimized versions of the algorithm to quantify improvements.

MIGLIORAMENTI FUTURI

Algoritmo nel tempo

Considerare il quadrato del segnale in modo da avere I picchi distribuiti sulla meta' della periodicita'.

Filtrare low-pass con filtri IIR potrebbe rendere piu' liscio il segnale (il FIR non produce effetti sualla forma d'onda) e si potrebbero quindi eliminare le irregolarita' che potrebbero confondere questo algoritmo.

Rilevamento dell'attacco

Utilizzare il Spectral Centroid in combinazione con l'attuale attack detection, una brusca variazione del centroid spettrale potrebbe significare un arpeggio sulla chitarra o un ribattuto sul piano.

FFT

La FFT deve essere migliorata perche' altrimenti non si possono riconoscere segnali generici. Per riconoscere frequenze piu' gravi si puo' fare uso di un Frequency Shift[1] (utilizzando la Trasformata di Hilbert), e per renedere piu' "smooth" il processo bisogna utilizzare una finestratura ed un overlap tra frame consecutivi.

Qui l'interpolazione parabolica puo' essere usata per stimare correttamente I picchi (massimi relativi), ma non abbiamo frequenze attese a priori.

Il massimo carico computazionale sopportabile definisce il limite alla lunghezza massima di analisi.

Osservare anche il comportamento delle fasi, sappiamo che la pulsazione e' la derivata dell'angolo, quindi monitorare come cambiano le fasi potrebbe dirci qualcosa.

Logica per chitarra

a questo successivo stadio di logica ci arrivano vari vettori:

uno con le frequenze riconosciute, uno con lo stato di certezza (exist[]), uno con la media dei primi 3 armonici secondo fourier, uno con la media dei primi 3 armonici secondo RTFI

sulla base delle energie e della certezza faccio una selezione di note (principalmente per non considerare quelle implausibili e di breve durata)

Frequency Matching

Studiare Cepstrum e Multiple Correlation.

L'idea e' di moltiplicare lo spettro corrente con quello precedente invece di fare la media (provare a mettere una finestratura, vedere gli effetti con Python prima).

> autocorrelazione per capire la presenza di alcuni candidati (RTFI) nello stram
>
> autocorrelazione significa prodotto scalare delle (FFT)

1sta qui /home/luciamarock/Documents/AudioAnalyzer/appunti/freq_shift

Prs_list	Ibanez_list
A110.wav	A110.wav
A220.wav	A220.wav
A440.wav	A440.wav
A880.wav	A880.wav
Ad116.wav	Ad116.wav
Ad233.wav	Ad233.wav
Ad466.wav	Ad466.wav
Ad932.wav	Ad932.wav
B123.wav	B123.wav
B247.wav	B247.wav
B494.wav	B494.wav
B988.wav	B988.wav
C1047.wav	C1047.wav
C130.wav	C130.wav
C261.wav	C261.wav
C523.wav	C523.wav
Cd1109.wav	Cd1109.wav
Cd138.wav	Cd138.wav
Cd277.wav	Cd277.wav
Cd554.wav	Cd554.wav
D1175.wav	D1175.wav
D146.wav	D146.wav
D294.wav	D294.wav
D587.wav	D587.wav
Dd1245.wav	Dd1245.wav
Dd155.wav	Dd155.wav
Dd311.wav	Dd311.wav
Dd622.wav	Dd622.wav
E165.wav	E1319.wav
E330.wav	E165.wav
E659.wav	E330.wav
E82.wav	E659.wav
F175.wav	E82.wav
F349.wav	F175.wav
F698.wav	F349.wav
F87.wav	F698.wav
Fd185.wav	F87.wav
Fd370.wav	Fd185.wav
Fd740.wav	Fd370.wav
Fd92.wav	Fd740.wav
G196.wav	Fd92.wav
G392.wav	G196.wav
G784.wav	G392.wav
G98.wav	G784.wav
Gd104.wav	G98.wav
Gd208.wav	Gd104.wav
Gd415.wav	Gd208.wav
Gd831.wav	Gd415.wav
	Gd831.wav

Pitch Detection Algorithm Plan

FFT-Based Pitch Detection:

Implement an FFT-based pitch detection algorithm.
Apply FFT to input signals for frequency analysis.
Determine peak frequencies in the FFT spectrum.

Frequency Shift for Low Frequencies:

Design a frequency shifting technique for targeting lower frequencies.
Apply the frequency shift to input signals.

Overlap for Accuracy:

Implement frame overlapping to capture transient features.
Optimize frame overlap size for accuracy and performance.

Parabolic Interpolation:

Develop parabolic interpolation method for precise frequency peak estimation.
Apply parabolic interpolation to identify frequency peaks.

Computational Load Consideration:

Implement dynamic window length adjustment based on computational load.
Optimize window length for efficient execution and accurate results.

Modulating Frequency:

Create a modulating frequency technique for frequency shifting.
Apply modulating frequency to enhance frequency component detection.

Phase Analysis:

Explore phase changes in input signals.
Analyze phase derivatives to identify harmonic transitions.

Theoretical Basis:

Establish a theoretical foundation linking angular velocity (omega) and phase derivatives.
Apply the theoretical basis to interpret phase changes in terms of frequency variations.
The proposed pitch detection algorithm plan combines established techniques and creative ideas to enhance accuracy and effectiveness. By implementing

these steps, the algorithm aims to provide reliable pitch detection results, especially for signals resembling membrane vibrations.

Time Approach Description

Load Waveform Data

The algorithm begins by loading waveform data from an external file.

Calculate Squared Values

The algorithm processes the waveform data in chunks of a specified size.

For each chunk, the squared values of the data points are calculated and stored.

Load Frequency Values

Frequency values corresponding to musical notes are loaded from an external file.

Calculate Samples per Period

The algorithm calculates the number of samples per period for each frequency.

These values represent the number of samples needed to complete one cycle of the waveform at each frequency.

Pitch Detection

The algorithm performs pitch detection by comparing squared values within each chunk.

For each frequency, the algorithm iterates over possible phase shifts within the chunk.

It calculates the average squared value over a period and identifies the maximum value.

The frequency with the highest detected squared value is considered the most prominent pitch.

Display Results

The detected pitch frequency is displayed as the output of the algorithm.

Future Enhancement (TODO)

The algorithm proposes a future enhancement by considering the square of signals for pitch detection.

This approach could potentially improve sensitivity to amplitude changes and waveform crossings.

Conclusion

The algorithm provides a basic pitch detection mechanism using squared values and waveform analysis.

Further exploration of signal processing techniques may enhance its accuracy and performance.

commenti in italiano

Logic

In the following logic stage, various vectors are received as input:
Frequency Recognized Vector: This vector contains the recognized frequencies.
Certainty State Vector (exist[]): This vector indicates the state of certainty for each frequency.
First 3 Harmonic Mean (Fourier): This vector contains the mean of the first three harmonics according to the Fourier analysis.
First 3 Harmonic Mean (RTFI): This vector contains the mean of the first three harmonics according to the Real-Time Frequency Identification (RTFI) analysis.
Based on the energy levels and the certainty state, a selection of notes is made. The primary purpose of this selection is to exclude implausible notes of short duration.

back to GitHub

```
# CONDIZIONE 1 basata sugli armonici
if (j<(len(data.T)-20) and vectnote[j]>0.0 and allowance > 0.0 and
vectnote[j+12]<vectnote[j]*Pshp*(1.0+Pthp) and vectnote[j+12]>vectnote[j]*Pshp*(1.0-Pthp) and
vectnote[j+19]<vectnote[j]*Pshs*(1.0+Pths) and vectnote[j+19]>vectnote[j]*Pshs*(1.0-Pths)):

# CONDIZIONE 2    basata sull'algoritmo di detection della waveform
elif (j<(len(data.T)-20) and existsum<5 and vectmono[j]>0.0 and allowance > 0.0 and
vectnote[j+12] < vectmono[j]*Pshp*(1.0+Pthp) and vectnote[j+12]>vectmono[j]*Pshp*(1.0-Pthp)):


# CONDIZIONI 3 e 4 per note acute

3

elif (j>27 and j<38 and vectnote[j]>0.0 and vectnote[j]>data[i,j-12]/170*1.15 and
vectnote[j]>data[i,j-19]/170*1.15 and allowance > 0.0 and vectnote[j+12]>vectnote[j]*0.75 and
vectnote[j+12]<vectnote[j]*1.25):


4

elif (j>37 and j<51 and vectnote[j]>0.0 and vectnote[j]>data[i,j-12]/170*1.15 and
vectnote[j]>data[i,j-19]/170*1.15 and allowance > 0.0 and (datafft[i,j]
+datafft[i,j+12]+datafft[i,j+19])>(datafft[i,j-12]+datafft[i,j]+datafft[i,j+7])*0.75 and (datafft[i,j]
+datafft[i,j+12]+datafft[i,j+19])>(datafft[i,j-19]+datafft[i,j-7]+datafft[i,j])*0.75):



# CONDIZIONE 5    quinte

elif(j>6 and j<39 and vectnote[j]>0.93*(vectnote[j-7]+vectnote[j+5])*0.5 and exist[j-7]>0 and
datafft[i,j+12]>0.66*datafft[i,j-7]):

# CONDIZIONE 6    terze

elif (j>15 and j<47 and ((exist[j-16]>0 and exist[j-4]>0 and exist[j+3]>0 and
vectnote[j]>0.93*(vectnote[j-16]+vectnote[j-4]+vectnote[j+3])/3.) or (exist[j-15]>0 and exist[j-3]>0
and exist[j+4]>0 and vectnote[j]>0.93*(vectnote[j-15]+vectnote[j-3]+vectnote[j+4])/3.))):
```

To integrate the frequency domain approach with the existing time domain approach, you can consider the following steps in the logic block of your algorithm:

1. **Initial Note Recognition using Time Domain Approach:**

- Implement the initial note recognition using the existing time domain approach. This would provide a quick identification of the first note played, leveraging the monophonic nature of the signal.

2. **Frequency Domain Analysis:**

- Apply the frequency domain approach to analyze the signal and obtain information such as the recognized frequencies and the mean of the first three harmonics according to Fourier analysis.

3. **Logic Integration:**

- Develop a logic block that integrates information from both time and frequency domain analyses. Consider the following strategies:

    a. **Priority Scheme:**
        - If the time domain analysis provides a clear and confident identification of the first note, prioritize that information.
    b. **Comparison and Verification:**
        - Compare the results from the time and frequency domain analyses. If they align, it adds confidence to the note identification. If there is a discrepancy, consider a verification step or prioritize the result with higher certainty.
    c. **Combining Strengths:**
        - Leverage the strengths of each approach. For example, if the time domain is better at quickly identifying the initial note, use it for quick recognition. If the frequency domain is better for sustained analysis or harmonic information, use it to refine and support the time domain results.

4. **Refinement and Feedback:**

- Implement a feedback loop to refine the results based on ongoing analysis and user feedback. This could involve iteratively adjusting the weights or priorities assigned to the time and frequency domain results.

5. **Thresholds and Certainty:**

- Establish thresholds for confidence or certainty levels from both approaches. Use these thresholds to determine when to trust one approach over the other.

6. **Output:**

- The final output could include information from both time and frequency domain analyses, potentially providing a more robust and accurate result.

Here's a simplified example of how you might structure the logic:

```
# Time Domain Approach
initial_note = recognize_initial_note(time_domain_signal)

# Frequency Domain Approach
frequencies, certainty, mean_harmonics = frequency_domain_analysis(frequency_domain_signal)

# Logic Integration
if initial_note.certainty > time_threshold:
    final_note = initial_note
else:
    if certainty > frequency_threshold:
        final_note = Note(frequencies, certainty, mean_harmonics)
    else:
        final_note = UndeterminedNote()

# Output
return final_note
```

Adjust the specific logic and parameters based on the characteristics of your signals and the performance of each analysis approach.

Signal early Periodicity Extraction

Edit New page

Jump to bottom

Luciano Ciamarone edited this page 19 minutes ago · 2 revisions

Fundamental Periodicity Extraction

Defining Parameters for Analysis

- `start` is the starting index for analysis.
- `nsamp` is the number of samples in a chunk.
- Initialize arrays `chunk` and `prog` to store squared data values and their corresponding indices.

```
start = 4668
nsamp = 256
chunk = [0] * (3 * nsamp)
prog = [0] * (3 * nsamp)
```

Extracting Periodicity Features

- Loop over the range of 3 times `nsamp` .
- Populate `chunk` with the square of audio data values starting from index `start` .
- Fill `prog` with indices corresponding to the data points.

```
for i in range(3 * nsamp):
    chunk[i] = pow(data[start + i], 2)
    prog[i] = i
```

Loading Note Frequencies and Calculating Samples

- Load note frequencies from the file "notes.txt".
- Calculate the number of samples per cycle for each note frequency.

```
note = genfromtxt("notes.txt")
samples = [0] * len(note)
for i in range(len(note)):
    samples[i] = int(round(44100. / note[i]))
```

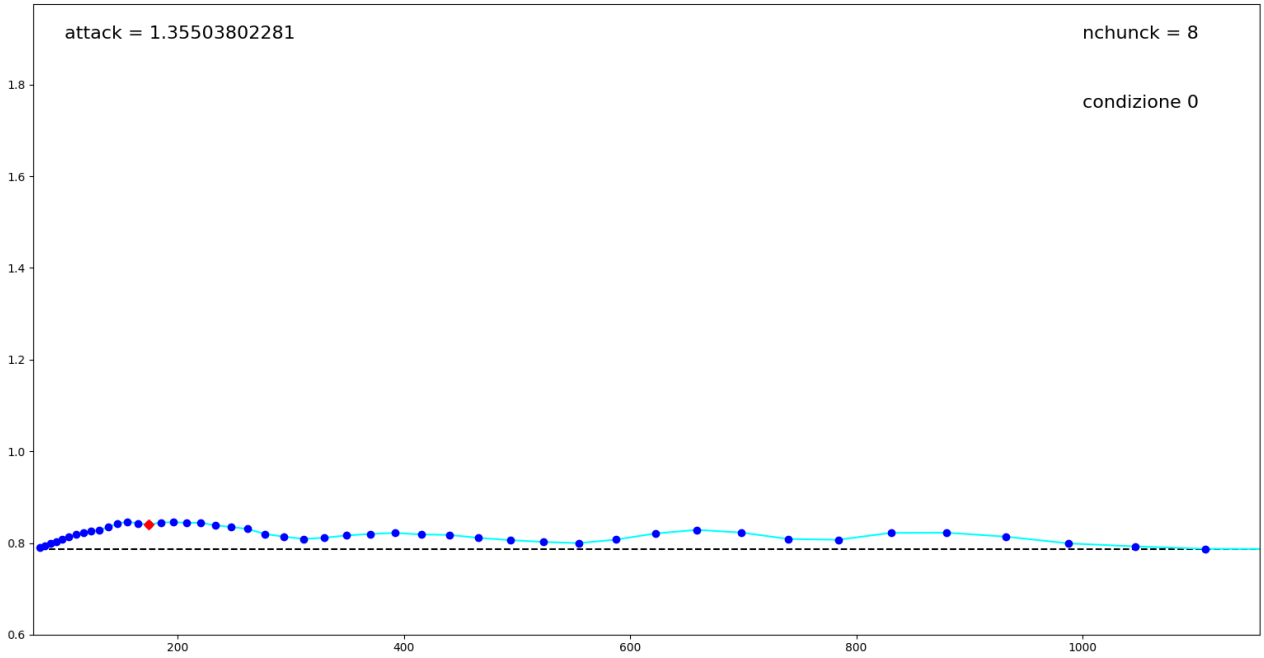
Fundamental Periodicity Detection

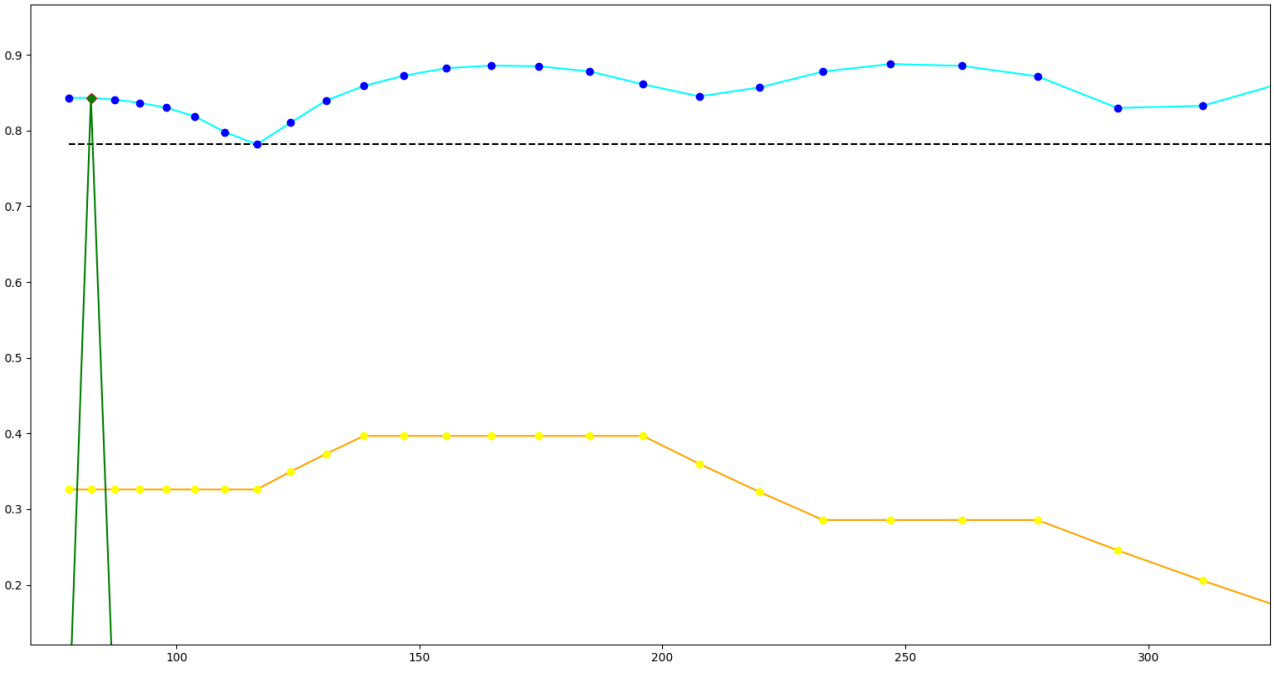
- Iterate through each note's sample count.
- Use a sliding window of size `samples[i]` to calculate an average value `capture` of squared data values.
- Track the maximum `capture` value within the sliding window.
- Detect the fundamental periodicity by finding the note frequency with the maximum `capture` value.

```
detect = 0
counter = 0
for i in range(len(samples)):
    passi = 3 * nsamp - samples[i] - 2
    maxi = 0.
    for j in range(passi):
        capture = (chunk[j] + chunk[j + 1] + chunk[j + 2] + chunk[j + samples[i]]
        if capture > maxi:
            maxi = capture
    if maxi > detect:
        detect = maxi
        counter = i
print(note[counter])
```

Comment on Squaring the Signals

- There's the possibility that squaring the signals might provide quicker results since it takes into account both the peak and trough of the waveform.





commenti in italiano

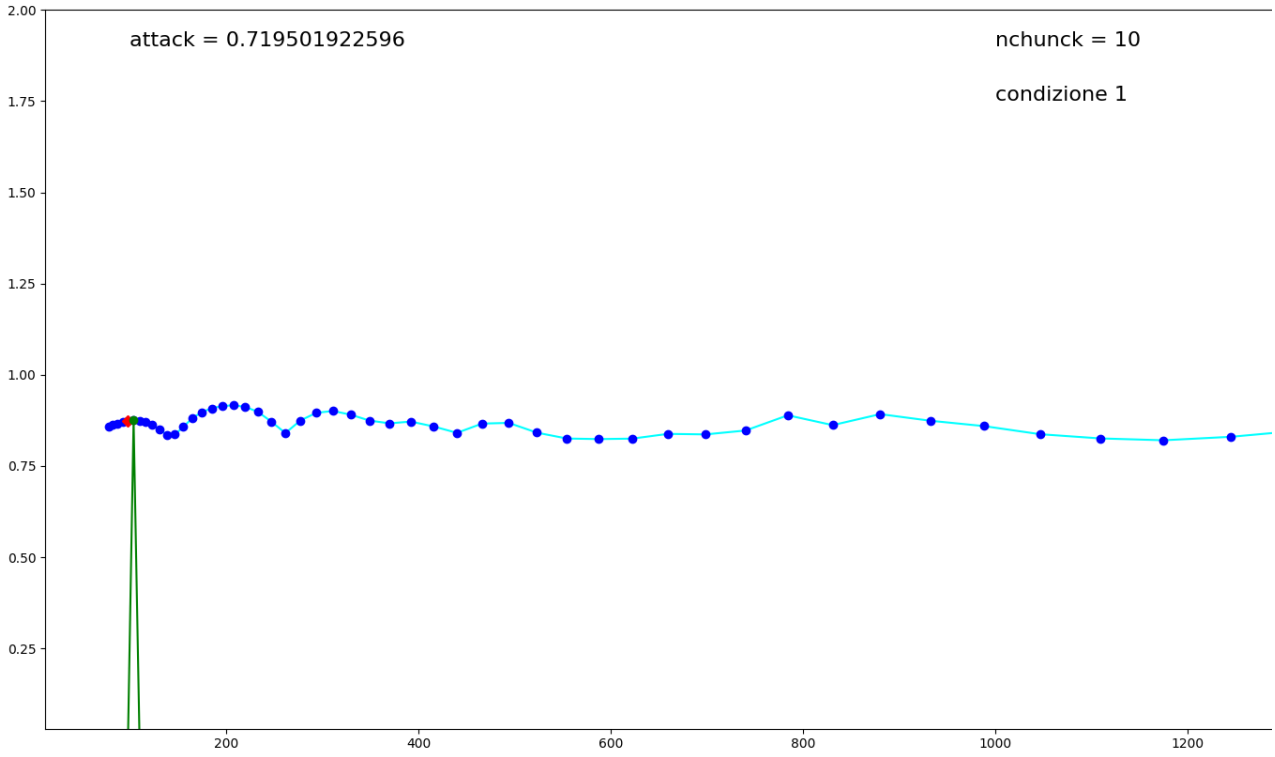
+ Add a custom sidebar

Clone this wiki locally

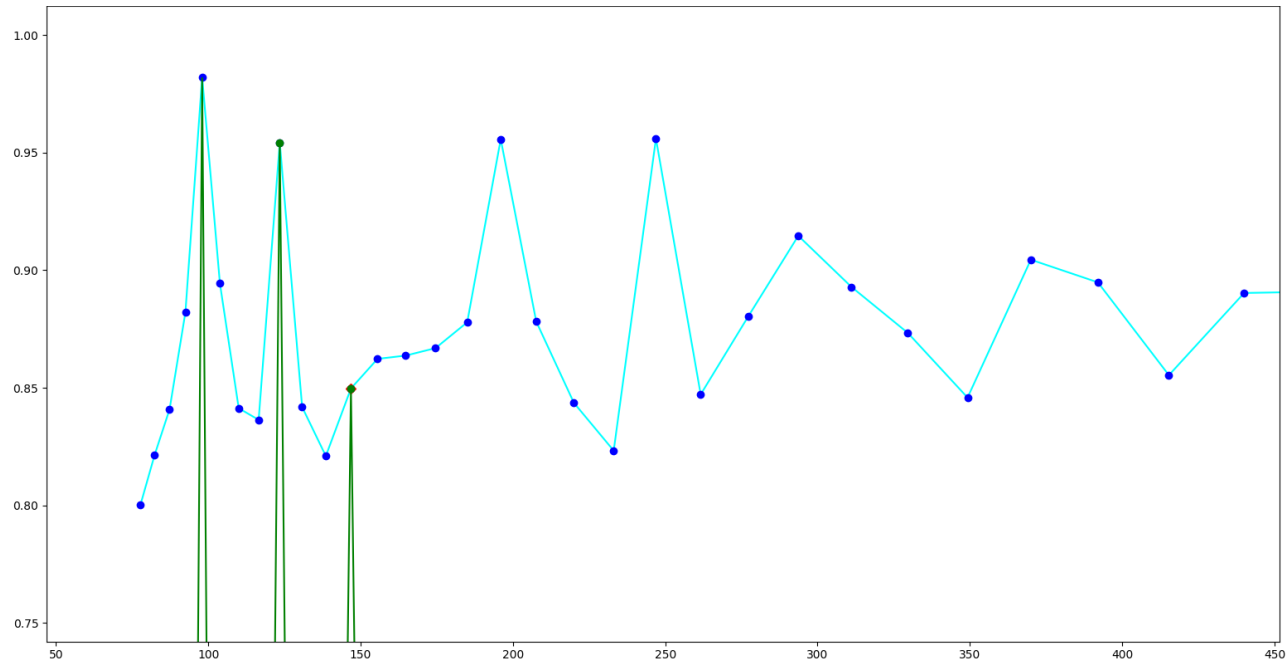
```
https://github.com/OnlinePianist/PolyphonicPitchDetection.wiki.git
```

Time domain algorithm

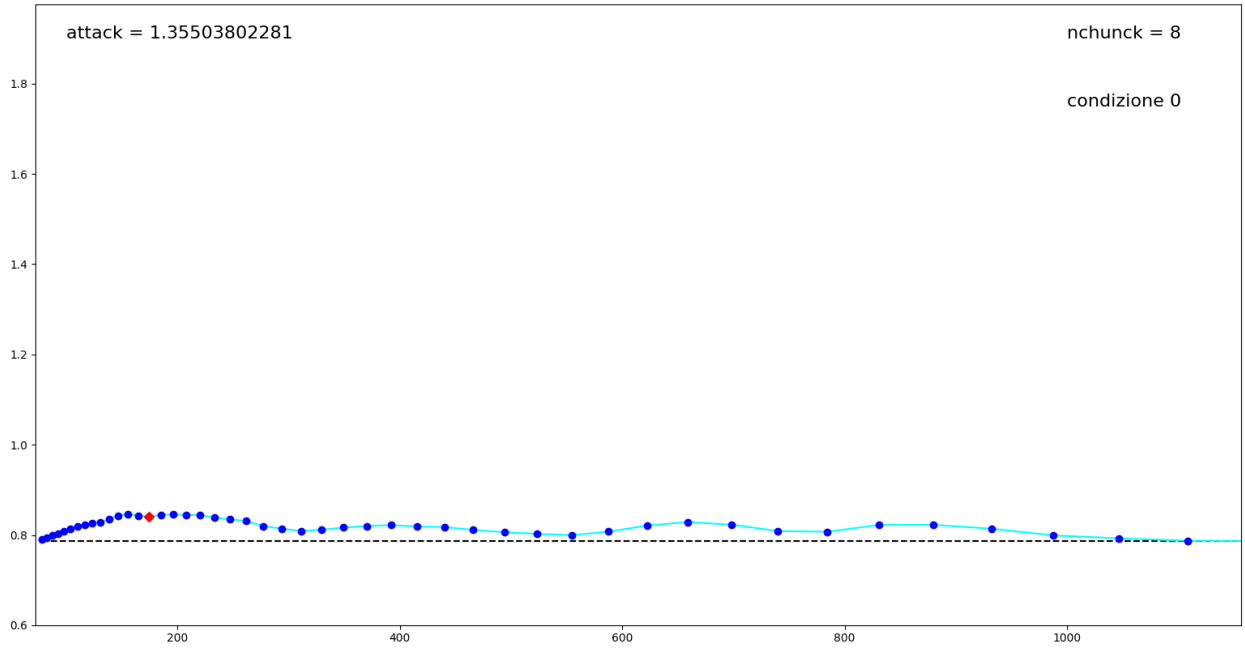
In the attached image, the detection of the "fraseG98triade" is shown. The logic prefers "condition 1," even though it results in a G# pitch. However, the algorithm over time correctly provides a G pitch. Trusting "condition 1" more than the time-based algorithm was a choice that could be reconsidered in the future. For instance, it could be suggested that if the first harmonic is around the pitch estimated by the monophonic (time-based) detection, then one should rely on the monophonic detection rather than "condition 1."



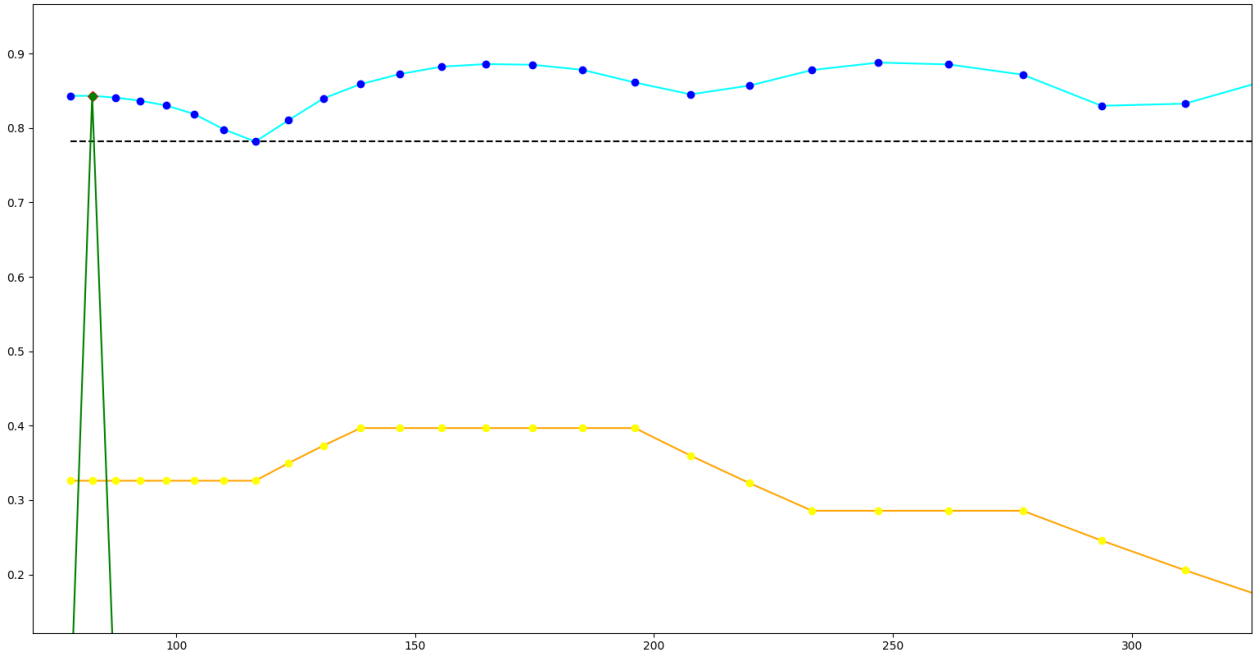
Regarding the "faseG98triade" file:
In the associated image, it can be observed that the time-based algorithm recognizes the third note of the triad before the resonator detects it. If its first harmonic exists, the algorithm considers it real ("condition 2").



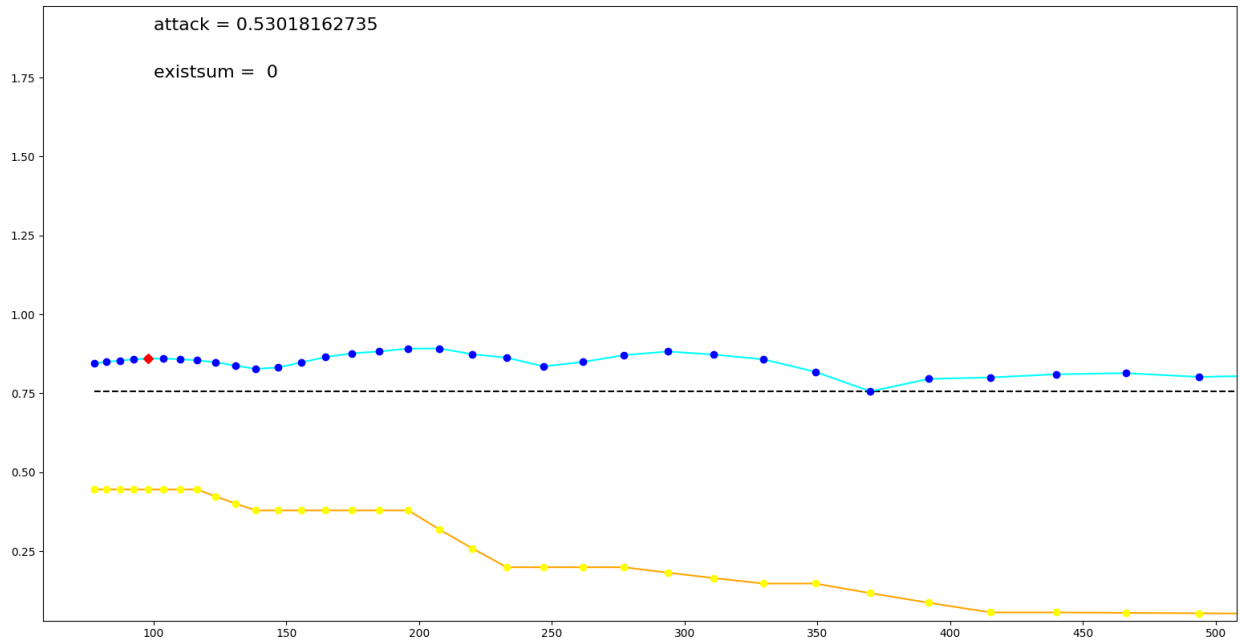
For two consecutive instances, the time-based algorithm provides the correct result, but the first harmonic was not identified by the resonators. Then, in the subsequent instant, this happens. Perhaps it can be stated that if the time-based algorithm detects the same note for two consecutive times (and "existsum" == 0), then that note should be considered as existing.



To enhance the effectiveness of this algorithm, it is possible to apply a low-pass filter to the waveform.

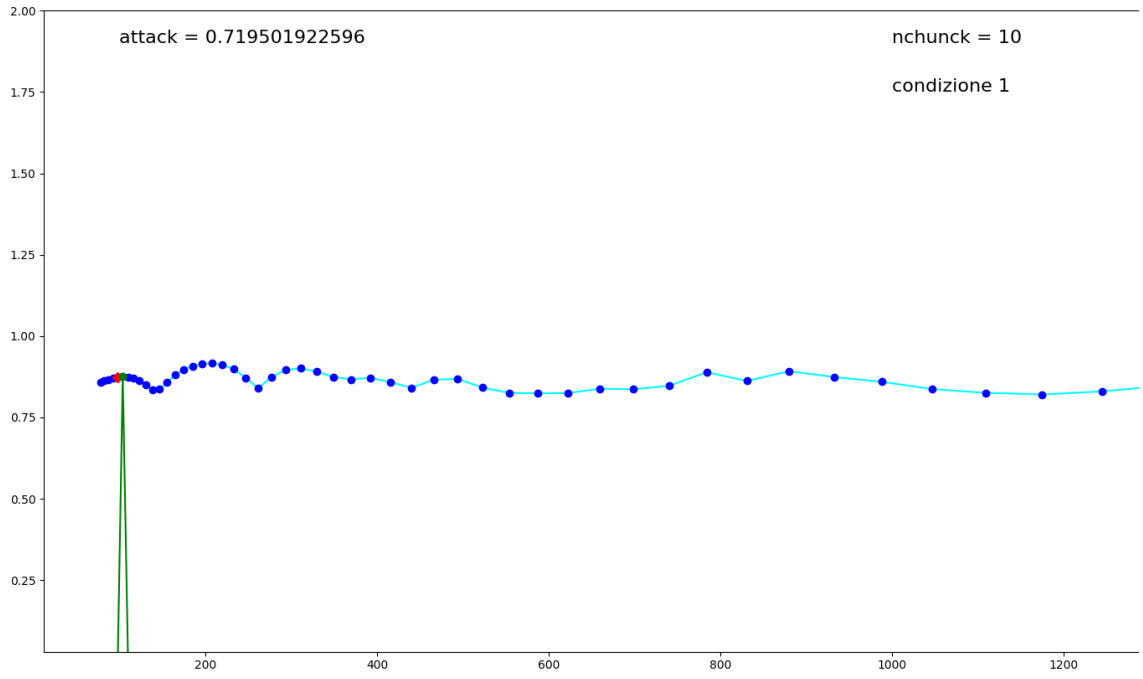


However, an IIR (Infinite Impulse Response) filter should be used instead of an FIR (Finite Impulse Response) filter because the FIR filter preserves the shape of the waveform. Therefore, if there are ripples that deceive the time-based algorithm, they would remain.

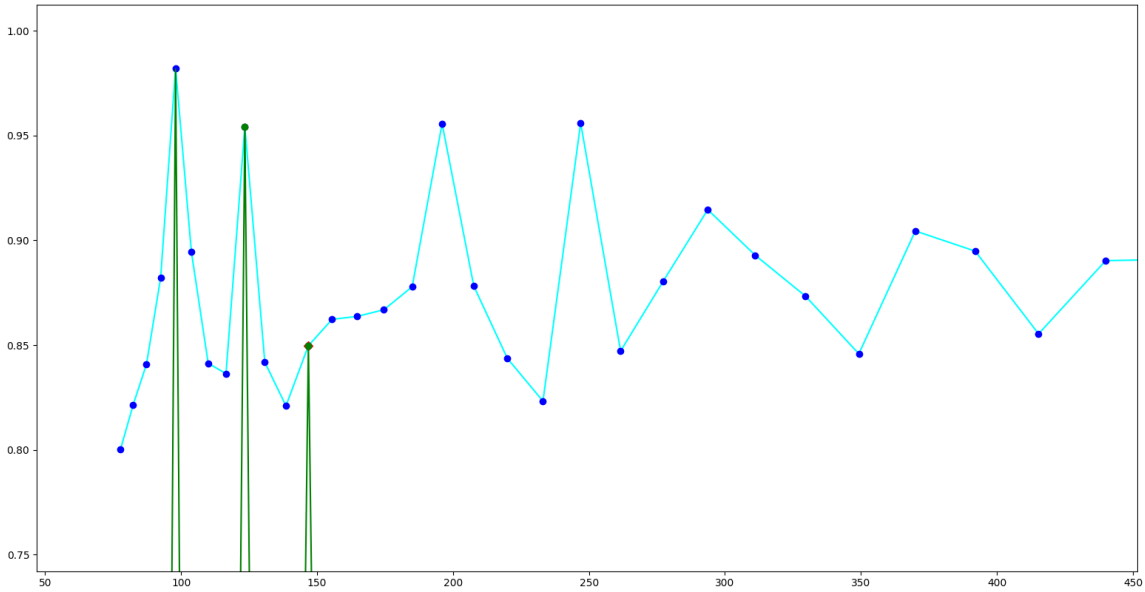


periodicity detection

nell'immagine allegata c'è la detection di fraseG98trieade, la logica preferisce la condizione 1 anche se essa fornisce il sol#
l'algoritmo nel tempo invece fornisce correttamente il sol; fidarmi della condiione 1 più che dell'algorimto nel tempo è stata una scelta che magari in futuro si può cambiare; ad esempio può dire che se il primo armonico si aggira intorno a quello stimato dalla detection monofonica (nel tempo) allora bisogna fidarsi della detection monofonica e non della condizione 1.

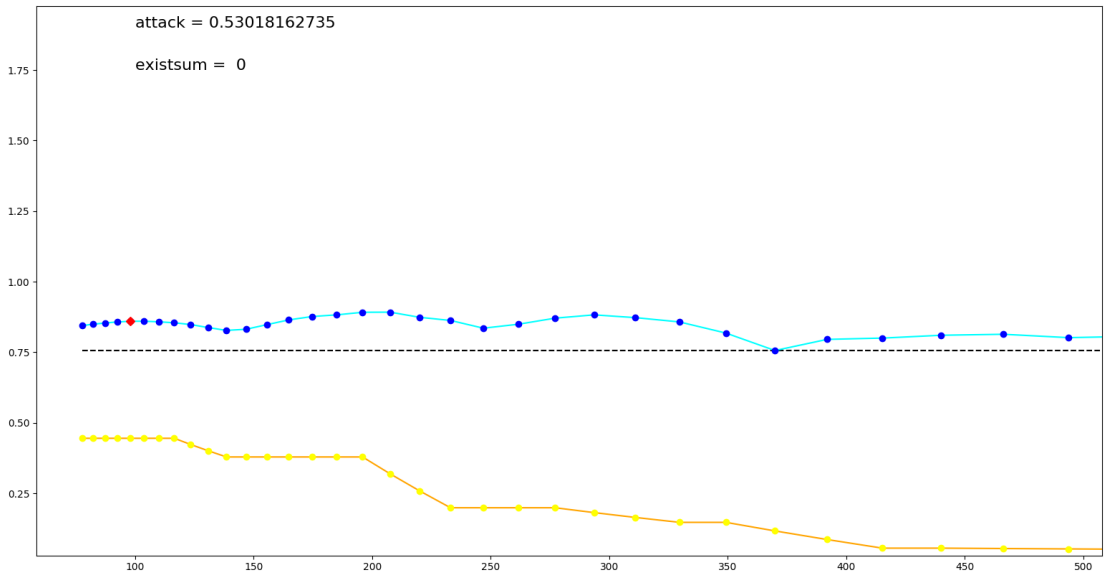


file faseG98triade:
nell'immagine associata si vede come l'algo nel tempo fa si che venga riconosciuta la terza nota della triade prima che il risuonatore la rilevi, se esste il suo primo armonico la considera realmente esistente (condizione 2)



per due istanti consecutivi l'algo nel tempo ha fornito il giusto risultato ma il primo armonico non era stato identificato dai risuonatori
poi all'istante sucessivo questo accade.
forse si puo' dire che se per due volte consecutive l'algo nel tempo rileva la stessa nota (e existsum == 0), allora quella nota va fatta esistere

per rendere piu efficace questo algoritmo si puo filtrare passa basso la forma d onda pero con un filtro IIR no con un FIR perche il FIR conserva lo shape
e quindi se ci sono delle increspature che ingannano l algoritmo nel tempo, queste rimangono



Back to Spunti

JamOrigin's pitch detection algorithm is a software that is used to analyze and detect the pitch of an audio signal in real-time. It works by analyzing the frequency content of the audio signal and identifying the dominant pitch, or the frequency with the highest amplitude. The algorithm then maps this pitch to a specific note on a musical scale, such as C, D, E, etc.

The first step in the process is to convert the audio signal into the frequency domain using a technique called a Fast Fourier Transform (FFT). This process breaks the audio signal into its individual frequency components. The algorithm then examines each frequency component and calculates the amplitude, or the strength, of each one.

Next, the algorithm uses a **pitch detection function** to identify the frequency component with the highest amplitude, which is considered to be the dominant pitch. This pitch detection function can be based on a variety of different techniques, such as the **autocorrelation method**, the **cepstral method**, or the **harmonic product spectrum method**.

Once the dominant pitch is identified, the algorithm maps it to a specific note on a musical scale. This is done by comparing the pitch to a pre-defined set of pitch values for each note. The algorithm then outputs the detected note and its corresponding pitch for use in a software like a guitar tuner, a MIDI instrument, or a music notation software.

JamOrigin's pitch detection algorithm is known for its high accuracy and low latency, making it suitable for real-time applications such as live performances and music production.

The pitch detection function is a key component of JamOrigin's pitch detection algorithm. It is used to identify the dominant pitch, or the frequency with the highest amplitude, in the audio signal. There are several different techniques that can be used as a pitch detection function, each with their own advantages and disadvantages.

One common technique is the **autocorrelation method**. This method calculates the similarity between a signal and a delayed version of itself. The pitch is then determined by finding the delay that produces the highest correlation.

Another technique is the **cepstral method**, which is based on the idea that the pitch period is present in the logarithm of the signal's power spectrum. The cepstral method involves taking the inverse Fourier transform of the logarithm of the power spectrum, and then finding the pitch period by finding the location of the peak in the resulting signal.

Harmonic Product Spectrum (**HPS**) method is also a popular technique where the algorithm multiplies the original audio signal with itself at different harmonic frequencies. The resulting signal is then used to find the pitch by looking for the frequency where all the harmonics align.

The choice of pitch detection function depends on the specific application and the desired trade-off between computational complexity, accuracy and latency.

JamOrigin's pitch detection algorithm is known for its high accuracy and low latency, making it suitable for real-time applications such as live performances and music production. It may use different techniques in different situations, depending on the desired result.

JamOrigin's pitch detection algorithm is polyphonic, meaning it is able to detect multiple pitches or notes in an audio signal simultaneously. In order to identify the real notes and distinguish them from harmonics, the algorithm uses a technique called pitch tracking.

Pitch tracking is the process of following the evolution of the pitches over time. The algorithm analyses the audio signal over a series of overlapping frames, each with a fixed duration. For each frame, it detects the pitches using the pitch detection function and then associates them with the pitches detected in the previous frame. This way it can track the evolution of each pitch over time, and decide which ones are real notes and which ones are harmonics.

The algorithm also uses additional information such as the amplitude and frequency of each pitch to help distinguish real notes from harmonics. For example, a real note is likely to have a higher amplitude and a more stable frequency than a harmonic.

Additionally, the algorithm can also use additional information such as the spectral envelope, the harmonic structure, and the pitch salience (how prominent the pitch is) to further improve the accuracy of the pitch tracking.

In summary, JamOrigin's pitch detection algorithm uses a combination of pitch tracking and additional information to distinguish real notes from harmonics in a polyphonic audio signal. This allows the algorithm to accurately detect multiple pitches or notes simultaneously in real-time, making it suitable for a wide range of applications such as live performances and music production.

The algorithm takes into account the relationship between the different pitches or notes in the audio signal.

In music, different notes have a specific harmonic relationship to each other. For example, a note played at a frequency of 440 Hz is an A note, and it is the 5th harmonic of an E note played at a frequency of 660 Hz. This relationship is known as harmonic series, and it is based on the fact that the frequencies of the notes are related by whole number ratios.

By using the harmonic structure, the algorithm can improve the accuracy of the pitch tracking by taking into account the relationship between the different pitches or notes. For example, if the algorithm detects two pitches that are related by a specific harmonic ratio, it is more likely that they are both real notes, rather than one being a harmonic of the other.

By using this information, the algorithm can make better decisions about which pitches to track and which ones to discard, resulting in a more accurate pitch tracking overall. Additionally, it can also use this information to help distinguish between different instruments or voices, which often have distinct harmonic structures.

It's important to note that the harmonic structure is one of the many information that the algorithm takes into account to improve the pitch tracking and that the exact implementation and weight of each information can vary depending on the specific algorithm and its intended application.

Machine learning can be used in a variety of ways to improve the accuracy and performance of pitch detection algorithms for music applications. Here are a few examples:

1. Supervised learning: One approach is to use supervised learning techniques to train a model to recognize the pitch of an audio signal. This can be done by providing the model with a dataset of audio signals and their corresponding pitch annotations. The model can then be used to predict the pitch of new audio signals.
2. Unsupervised learning: Another approach is to use unsupervised learning techniques to discover the underlying structure of the audio signals. For example, clustering algorithms

can be used to group similar audio signals together, and then the pitch of each group can be estimated based on the average pitch of the audio signals in the group.

3. Deep Learning: With the increasing popularity of deep learning, convolutional neural networks (CNN) and recurrent neural networks (RNN) are being used to improve the performance of pitch detection algorithms. These models can learn to extract features from audio signals in an unsupervised way, and then use these features to predict the pitch.
4. Adaptive filtering: Machine learning can also be used to adapt the parameters of a pitch detection algorithm in real-time, based on the characteristics of the input audio signal. For example, a pitch detection algorithm can use an adaptive filter to adjust its sensitivity to different frequencies or to different types of audio signals.
5. Ensemble learning: Another approach is to use ensemble learning techniques, where multiple models are trained and combined to improve the overall performance of the algorithm. This can be done by combining the predictions of multiple models or by training a meta-model that can learn to adapt to different scenarios.

Overall, machine learning can be used to improve the accuracy and performance of pitch detection algorithms in various ways, depending on the specific application and the desired trade-off between computational complexity, accuracy, and latency.

here are a few examples of how deep learning techniques can be used to extract features from audio signals and then use these features to predict the pitch:

1. Spectrogram analysis: One approach is to use a convolutional neural network (CNN) to analyze the spectrogram of an audio signal. A spectrogram is a visual representation of the frequencies present in an audio signal over time. The CNN can learn to extract features from the spectrogram, such as the shape and location of the frequency peaks, and then use these features to predict the pitch.
2. Time-domain analysis: Another approach is to use a recurrent neural network (RNN) to analyze the time-domain signal directly. The RNN can learn to extract features such as the amplitude and frequency of the audio signal over time, and then use these features to predict the pitch.
3. Cepstral analysis: A specific example of time-domain analysis is the cepstral analysis, where the input to the RNN is the cepstral coefficients of the audio signal. Cepstral coefficients are obtained by taking the logarithm of the power spectrum of the audio signal and then taking the inverse Fourier Transform of the result. This representation is particularly useful for pitch detection as it emphasizes the pitch periodicity.
4. Pitch salience function: Another approach is to use a pitch salience function, a measure of the prominence of the pitch in the audio signal. A deep neural network can learn to extract features related to the pitch salience and use them to predict the pitch.
5. Hybrid models: It is also possible to combine these different approaches by using hybrid models, where multiple features are extracted using different techniques and then combined to make a final prediction. For example, a model can extract the spectral and temporal features of the audio signal and then combine them to predict the pitch.

It's important to note that these are just examples and that the specific implementation of the deep learning algorithm can vary depending on the specific application and the desired trade-off between computational complexity, accuracy, and latency. Additionally, the performance of these models also depend on the quality and quantity of the data that is used to train them.