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]))
```

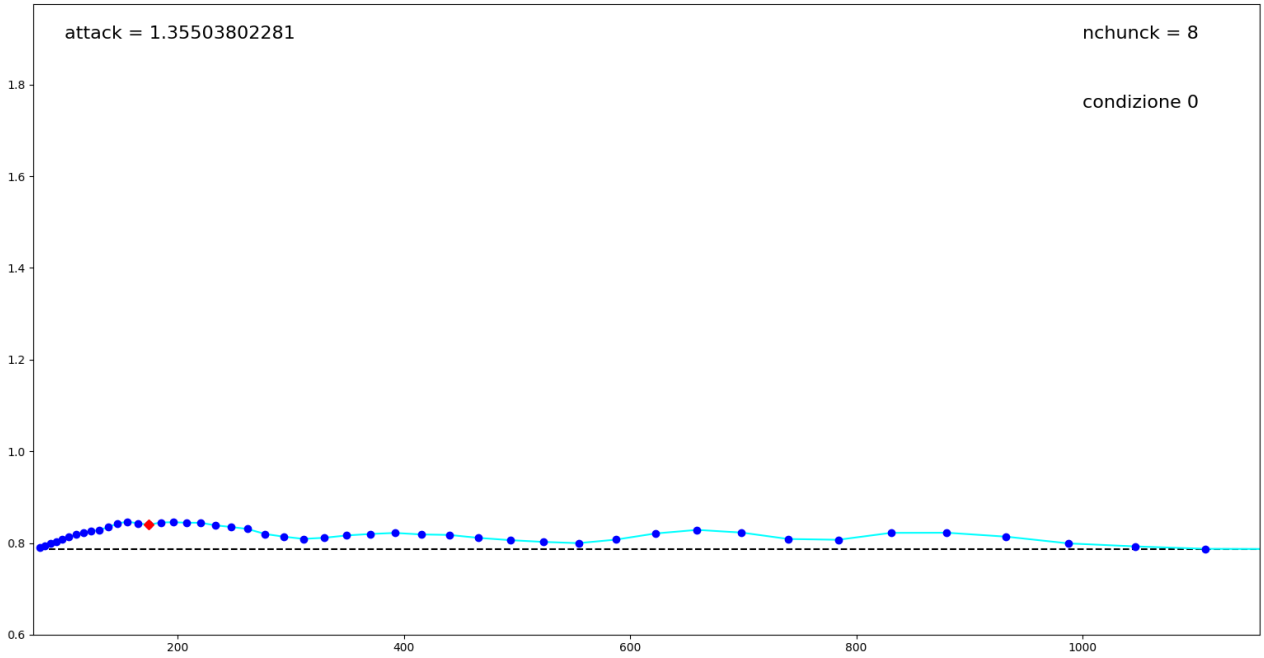
Fundamental Periodicity Detection

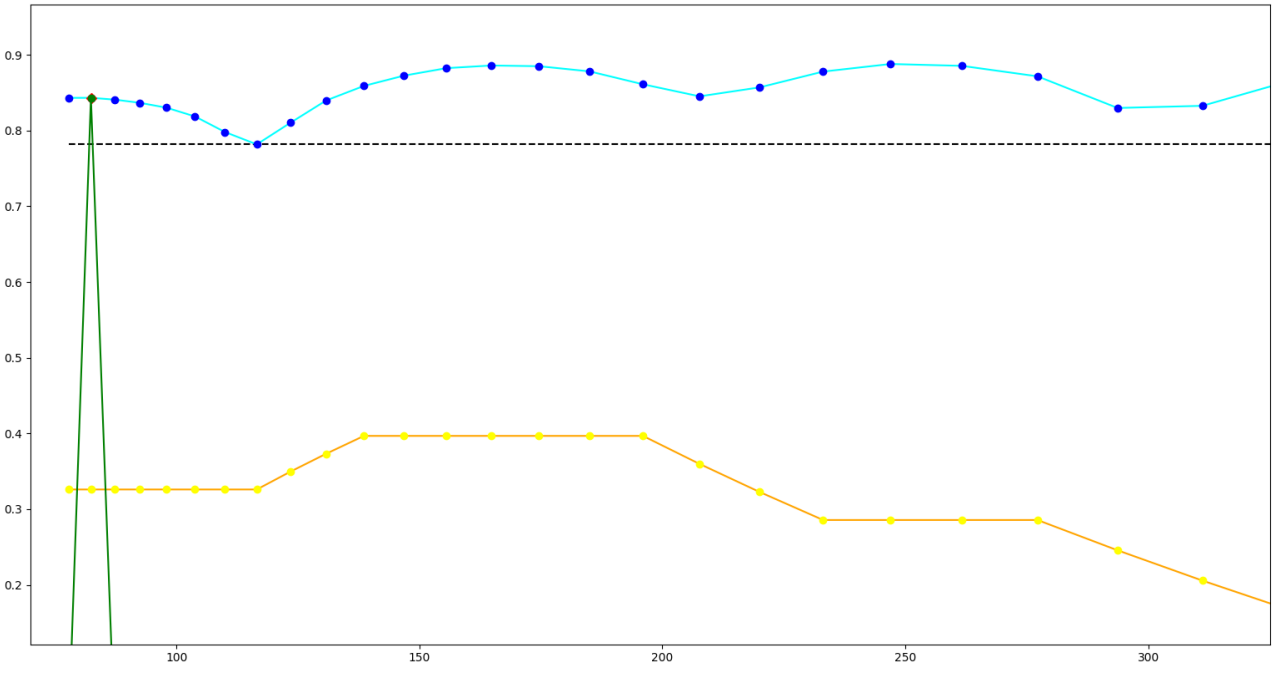
- Iterate through each note's sample count.
- Use a sliding window of size `samples[i]` to calculate an average value `capture` of squared data values.
- Track the maximum `capture` value within the sliding window.
- Detect the fundamental periodicity by finding the note frequency with the maximum `capture`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

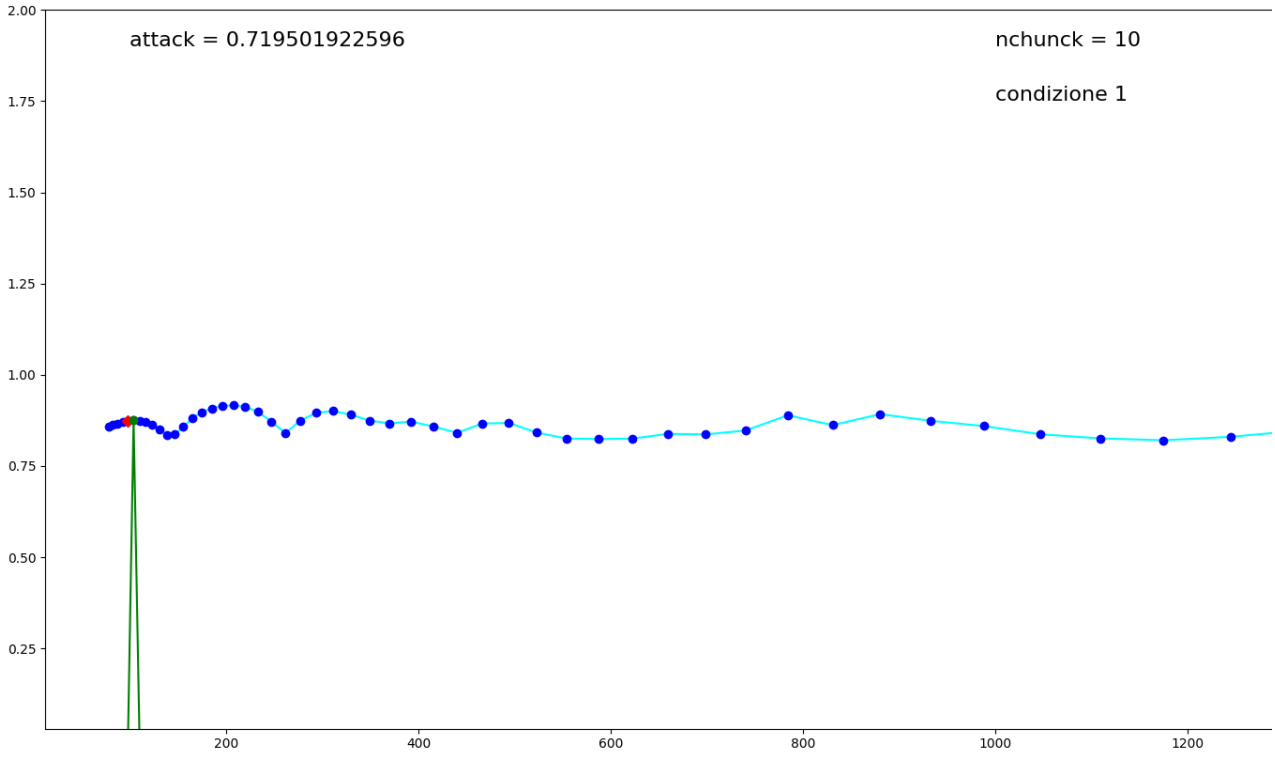
+ Add a custom sidebar

Clone this wiki locally

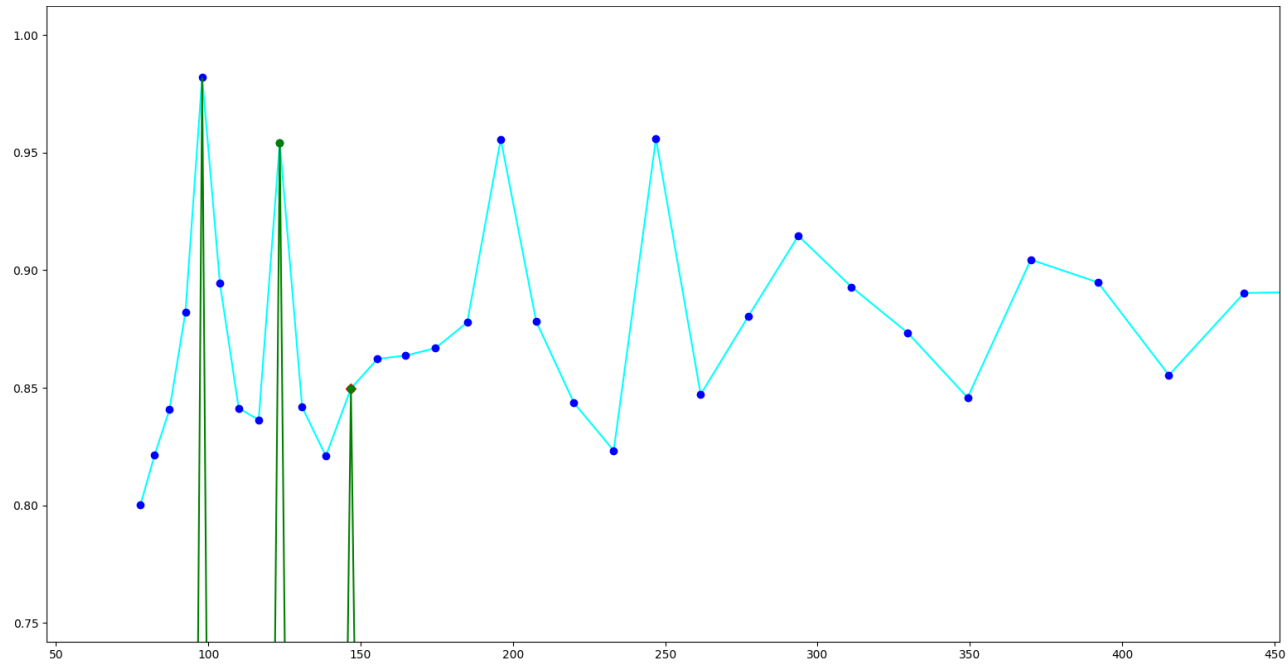
```
https://github.com/OnlinePianist/PolyphonicPitchDetection.wiki.git
```

Time domain algorithm

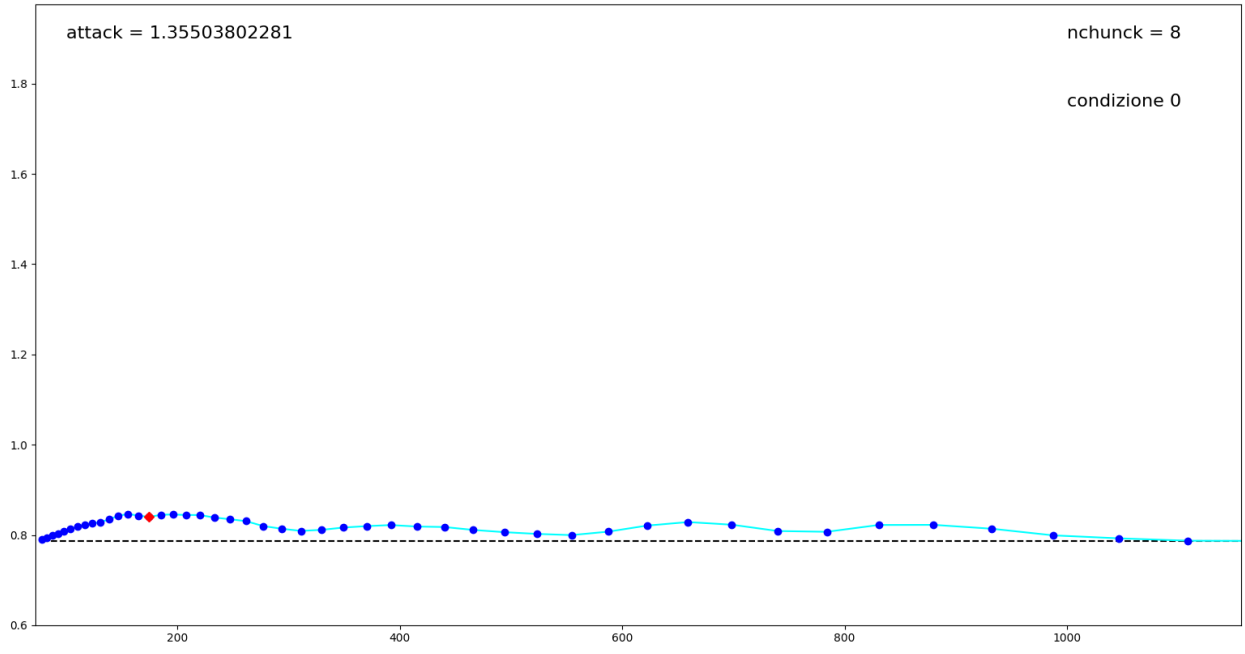
In the attached image, the detection of the "fraseG98triade" is shown. The logic prefers "condition 1," even though it results in a G# pitch. However, the algorithm over time correctly provides a G pitch. Trusting "condition 1" more than the time-based algorithm was a choice that could be reconsidered 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."



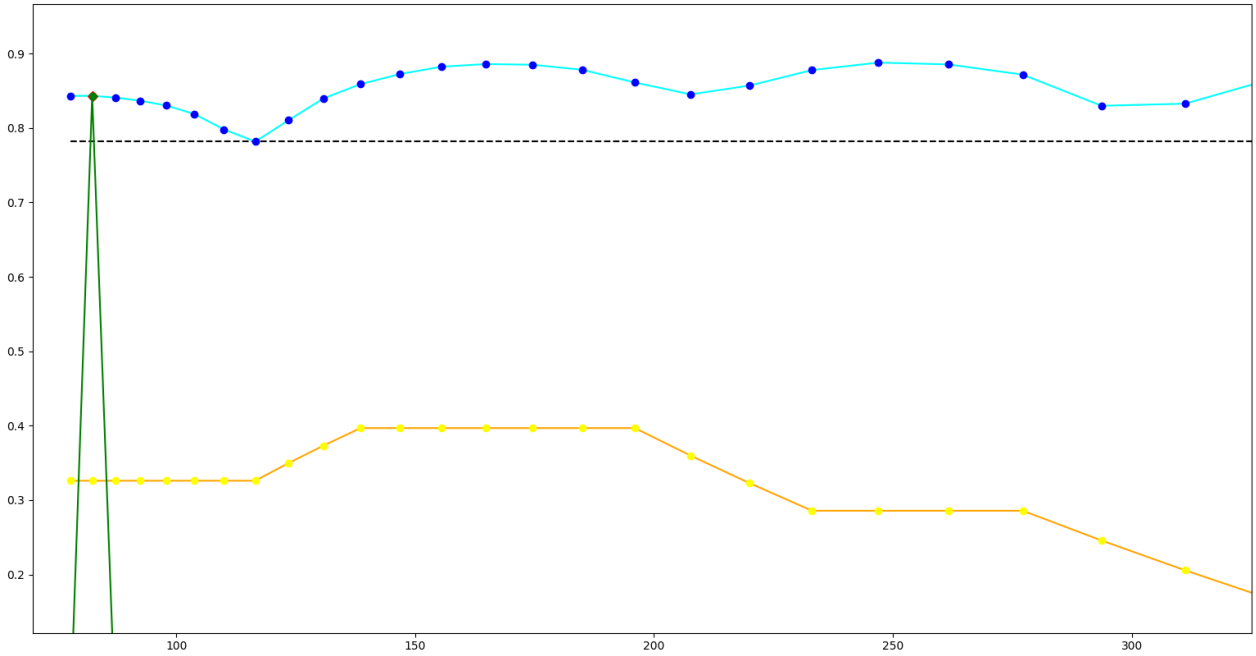
Regarding the "faseG98triade" file:
In the associated image, it can be observed that the time-based algorithm recognizes the third note of the triad before the resonator detects it. If its first harmonic exists, the algorithm considers it real ("condition 2").



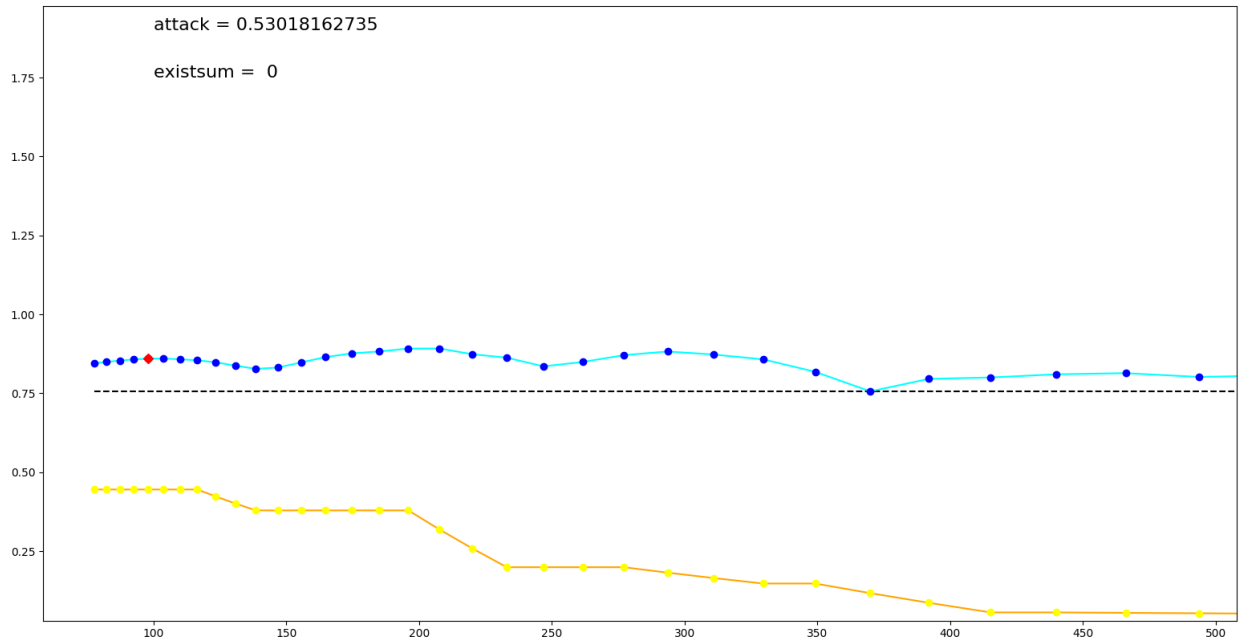
For two consecutive instances, the time-based algorithm provides the correct result, but the first harmonic was not identified by the resonators. Then, in the subsequent instant, this happens. Perhaps it can be stated that if the time-based algorithm detects the same note for two consecutive times (and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



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.