Detection of Sustain Points in Live Recorded Pitched Samples Using Frequency Estimation

Spencer Soodek
Northeastern University
Boston, US
spencersoodek@gmail.com

ABSTRACT

This paper describes a sampler using MaxMSP and JavaScript, that takes audio and MIDI as inputs. The sampler detects and stores the last audio event performed and allows the user to trigger a pitch shifted version of the note using MIDI input, pitch shifted by an interval relating to the distance between the input MIDI note and C4. If the audio event is a pitched sound, the sampler detects loop points in the sample, and allows the user to sustain the sample using the loop points until the end of the MIDI event. Loop points for each sample are determined using separation of the amplitude envelope into individual bands, zero crossings, and frequency estimation to predict appropriate lengths of looped sections.

1. INTRODUCTION

Looping audio is a common practice used in sampling to extend the duration of an audio sample with a repeating period. Looping can be used to continue a rhythmic section within a sample, or within a periodic recording with a loop of a smaller duration, to sustain a note within a sample. Figure 1 contains an example of a periodic recording with a section that can be looped to sustain the recording on feedback. Various audio file types allow loop points to be specified within the file to allow the loop to be followed during playback. Sampling using loops is commonly used with recordings of pitched instruments or synthesizer patches to allow the user to play a version of the instrument without the actual instrument present [1].

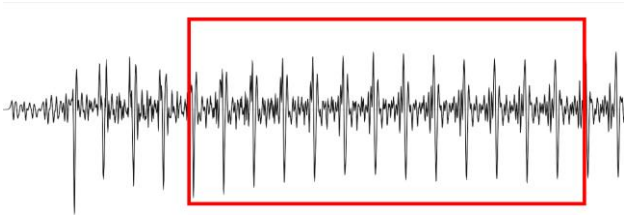

Figure 1. An example of a recording with a loopable region. On playback, the selected region can be looped before playing the end of the recording to extend the length of the recording.

Looping in sampling is best suited for instruments pitched instruments without modulation. Effects such as vibrato, tremolo, amplitude modulation, or filter modulation are difficult to capture within a loop, as the loop expects a signal with similar periods. When sampling instruments such as a synthesizer where these modulation effects can be adjusted, removing the effects in the original sound source, and applying them after sampling is recommended [1].

This project was inspired by working with orchestral libraries in instrument samplers such as Kontakt, which contain pitched recordings of individual instruments with loop points, allowing the user to be hold notes for different durations than that which the samples were originally recorded. The goal for this project is to create a sampler using MaxMSP that detects a valid unidirectional loop within a live recorded periodic sound, allowing the user to playback the sound with a loop automatically without manually specifying the bounds of the loop. This would allow a performer to create improvised recordings live and play them back as an instrument, assuming that the recordings come from a periodic source.

2. SAMPLING USING MAXMSP

This project is a sampler designed in MaxMSP, which records audio input, and allows the user to play back the recorded audio, pitch shifted using MIDI. Figure 2 contains a diagram of the signal flow for this sampler. The sampler uses two buffers, allowing uninterrupted playback of a previous sample while recording a new sample. At any given time, one of the buffers is “active”, while the other is armed for recording. The active buffer contains the sample that is played back if the user inputs a MIDI event, the recording buffer is recorded to if an audio event occurs.

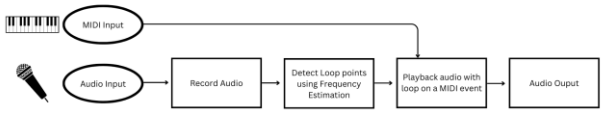

Figure 2. A diagram of the signal flow for this sampler.

Audio events are detected when the microphone detects a signal with an amplitude greater than 0.1. When such an audio event is detected, the buffer that is armed to record starts recording.

To detect the end of an audio event, there is a counter that increments every 80 milliseconds using the metro object in MaxMSP. Every time a sample with an amplitude greater than 0.1 is detected, the counter resets to 0. If the counter reaches 3, detecting that 160 ms have passed with no audio input greater than 0.1, the audio event is determined to be over, and the recording buffer stops

recording. When the recording stops, the recording buffer and the active buffer are swapped, so the next recording will record to what was previously the active buffer.

For playback, the sampler uses two MaxMSP groove~ objects. The groove~ object allows for variable playback speed on playback, allowing the user to shift the pitch by changing the speed based on the midi note performed, and the specification of loop points. The groove~ object that is played on a MIDI event is dependent on the buffer that is considered active at the time. The speed used for the audio signal in the groove~ object is determined by the frequency associated with the MIDI event performed in Hz, divided by 261.7, the frequency for middle C. The resulting frequency is used as the value for a sig~ object feeding into the active groove~ object. The performed pitch corresponds to the pitch of the original recording, shifted by the distance of the MIDI note to middle C. Middle C was chosen as the location to pitch shift around, rather than placing the recording on the key with its corresponding frequency, so the recording at its original frequency is always in an identifiable location, regardless of whether the user knows the pitch of the recording . A MIDI note-on event sends a loop 1 message to the groove~ object, and a MIDI note-off event sends a loop 0 message, allowing the sample to be looped for the duration of the MIDI event, then to stop looping when the MIDI event ends. The playback section is encapsulated within a patcher accessed using the poly~ object, allowing multiple notes to be performed at the same time.

Since recording does not start until a signal with an amplitude above 0.1 is detected, the playback requires a fade in to prevent pops and clicks due to a sudden change in amplitude. To account for this a midi on event triggers a line~ object, ramping from 0 to 1 over 20 milliseconds, and multiplies this signal by the output of the groove~ object.

3. LOOP POINT DETECTION

After a pitched audio event is recorded to a buffer, its loop points are detected. Valid loop points are based on the following criteria. Both the start and end points must be within the decay or sustain region of the amplitude envelope, after the initial attack. The first and second loop point must be spaced an integer multiple of one period apart to prevent phase shifting at each repetition of the loop. Phase shifting is problematic in a loop, as the looped section would be repeated at a frequency unrelated than the fundamental. first and second loop points must also occur at samples with equal amplitudes to prevent pops or clicks from sudden changes in amplitude at each repetition [2]. The current implementation of this sampler assumes that the audio event contains a relatively short attack, and a long decay which does not change pitch, similar to a plucked or bowed string, or a hammered piano.

The first loop point in the sample must occur after the initial attack. To find a suitable starting point located after the initial attack, using JavaScript within MaxMSP, the recording is split into non-overlapping windows with a width of 400 samples. The max amplitude within each window is compared, and the window with the greatest amplitude is determined to contain the attack. The start of the following window is determined to be a suitable spot to start looking for loop points.

Zero crossings are detected starting from the first sample in this second window, through the end of the recording. Zero crossings are suitable candidates for loop points, as they are locations within the period that are known contain the same amplitude at each repetition, as opposed to the peaks of a period which decay over time. Loop points must also occur at points with similar slopes to prevent pops or clicks due to sudden changes in slope [2]. To account for this, only zero crossings with negative slopes are detected. The first zero crossing following the end of the attack window is determined as the first loop point. The second loop point will also occur at a loop point, but first the distance between loop points must be calculated.

4. FREQUENCY DETECTION

The second loop point is positioned at a zero crossing located a distance of an integer multiplication of the period of the fundamental frequency from the first loop point. To calculate the fundamental frequency, the sampler uses a JavaScript implementation of the YIN algorithm proposed in [3]. The YIN name derives from the yin and yang, alluding to the combination of autocorrelation with cancellation at fundamental frequencies achieved through a difference equation. The YIN algorithm is chosen over pure autocorrelation for detecting frequency as it is less prone to errors when working with a periodic signal with changing amplitudes.

The YIN algorithm acts similarly to the autocorrelation method, by comparing the signal to its shifted self, using various lag times to find the shift that yields a similar signal in phase. The autocorrelation method compares the values of the signal to a shifted signal directly by adding the shifted signal to the original signal at the various lag times and searching for peaks. The autocorrelation method is prone to errors in a signal with imperfect periodicity due to changing amplitude, as the changing amplitude values between periods cause peaks to be less clear. To address this, rather than using direct comparison of the original signal and the shifted signal, the YIN algorithm uses a cumulative mean normalized difference function (1), where t represents the first sample within the recording, W represents the last sample, j represents the current sample being read, and r represents the length of the lag in samples between the original signal and the lagged signal. For each lag value, the value is found by subtracting the lagged amplitude of the signal from the original amplitude of the signal and squaring the difference, minimizing the effect caused by changing amplitudes.

$$\sum_{t+1}^{W}\left(x_j - x_{j+r}\right)^2 = 0 \qquad (1)$$

The result are values for each lagged time value which approach zero at possible lags that correspond to the fundamental frequency. This function minimizes the effect of changes of amplitudes between periods that occur in autocorrelation.

The YIN algorithm then implements a threshold for the difference between the cumulative mean normalized difference of the original signal and the lagged signal. Errors with choosing higher-order peaks occur can without the threshold, causing the returned frequency to correspond to a lower frequency or octave. The YIN algorithm uses a threshold to choose the lag time associated with the first difference local minimum below the threshold. If no local minimum below the threshold is found, the first lag time is chosen. The frequency returned from the YIN algorithm is the frequency associated with the lag time.

The JavaScript implementation of the YIN algorithm in this sampler uses the first loop point as the starting of the signal in the algorithm and lags the shifted signal between 1 and 1000 samples, which is sufficient the YIN algorithm to detect most fundamental frequencies in common sample rates. Delaying the signal for the entire length of the sample is inefficient for a periodic sample and cannot be run within MaxMSP fast enough to be usable for the intended purpose of creating loop points for a live sampler. Delaying the signal for the entire length of the sample is also unnecessary, as assuming the sample is periodic for the entire duration of the sample, at common sample rates of 44.1 KHz and 48 KHz, 1000 samples is sufficient to capture most frequencies within the range of human hearing.

Using the frequency returned from the YIN algorithm, the length of one period can be found dividing the sample rate by the period. To find the second loop point, an integer multiplication of the period length is added to the sample index of the first loop point, and the closest zero crossing is found. Both loop points have to be converted into milliseconds to be used with the groove~ object in MaxMSP. This can be achieved by dividing the sample index by the sample rate and multiplying by 1000.

5. CONCLUSION

In this project I have described a method for detecting loop regions in periodic samples using frequency estimation. Within the sampler described in this paper, this can be used in a live performance to record sustained textures and play them as an instrument in multiple pitches. This method of loop detection using frequency estimation can also be used outside of this project to simplify the process of selecting loops in periodic audio recordings, if the frequency of the original recording is unknown.

6. FURTHER DEVELOPMENT

This sampler is currently best suited for periodic sounds with short attacks like a plucked or bowed string or hammered piano. The audio input is assumed to have a short attack, with the periodic loop occurring after the attack. If the peak of the amplitude is towards the end of the recorded audio, the range to select loop points from is limited. Further development of this sampler would involve amplitude analysis of the entire sample to find sections that are periodic while minimizing the changes in amplitudes at the peak of each period.

This sampler also assumes that the audio signal that it receives as input is a periodic signal. The YIN algorithm that the sampler uses for frequency detection does not have a way of detecting whether a signal is entirely aperiodic, as it chooses the first difference value with a dip if there are no dips in difference values with a dip below the threshold. In the case of a percussive signal such as a drum which is aperiodic, the sampler still attempts to find a fundamental frequency and loop points which do not exist.

Currently the sampler uses the YIN algorithm to calculate the frequency and period length of the periodic signal proposed in [3]. The YIN algorithm selects a probable frequency based on a static threshold and outputs a single fundamental frequency estimation, which can result in errors in pitch estimation if the threshold is set incorrectly. The pYIN algorithm described in [4] assigns probabilities to multiple fundamental pitch candidates and uses a Hidden Markov Model to determine which candidate is most likely the correct pitch. A Hidden Markov Model selects a singular value from a list of probable values with associated probabilities based on previously selected values. In a recording where pitch changes over time, and multiple frames are used to track frequency changes, the pYIN algorithm is more effective than the pure YIN algorithm, as it prevents octave errors by using previous the frequencies calculated in previous frames to inform the selection of the current frequency. In this case, where the recording is assumed to be periodic and we are only calculating one frequency for the entire recording, this is not necessary as there are no previous frames to use to infer a current frame. However, if we were to expand this project to accommodate recordings that are not periodic for the entire duration, but have sections within the recording that are periodic, the pYIN algorithm would be more effective than the pure YIN algorithm.

REFERENCES

[1] C. Meyer and B. Aspromonte, “The art of looping (music technology, Dec 1987),” Music Technology, https://www.muzines.co.uk/articles/the-art-of-looping/2146 (accessed Apr. 8, 2024).

[2] C. Meyer, “The art of looping (music technology, Feb 1988),” Music Technology, https://www.muzines.co.uk/articles/the-art-of-looping/2246 (accessed Apr. 8, 2024).

[3] A. de Cheveigné and H. Kawahara, “Yin, a fundamental frequency estimator for speech and music,” The Journal of the Acoustical Society of America, vol. 111, no. 4, pp. 1917–1930, Apr. 2002. doi:10.1121/1.1458024

[4] M. Mauch and S. Dixon, "PYIN: A fundamental frequency estimator using probabilistic threshold distributions," 2014 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), Florence, Italy, 2014, pp. 659-663, doi: 10.1109/ICASSP.2014.6853678.