Final Project

MAX/MSP Looper and automatic drum player

MUMT 306: Music Audio Computing I

Autumn 2020
Schulich School of Music
McGill University

Last updated : December 10, 2020

Siméon **Le Grand** 260999667

Contents

1 Objectives

The objectives of this patch are fairly straight forward. The main goal is to be able to record a loop of any audio signal and add drums on top of that. The drum are to be played automatically and in sync with the rest of the loop. This implies that there must be a tempo detection algorithm that allows the drums to sync up with the loop. The recording process of the loop is to be the same as is typical of looper pedals that guitarists use. Meaning you tap once with your foot on the pedal, the recording starts, you tap another time and the recording ends. Also, when playing the loop over a long period of time, there should be no drift between the loop and the drums. Meaning that the drums and loop should still be in sync even if they have been playing for a long time. This is to be implemented by using the software MAX/MSP.

2 MAX/MSP Patch

2.1 Overview

The MAX/MSP patch can be broken into multiple components. The first section is the trigger of the looper. This is where the audio loop is told to start and stop recording. First, a trigger detection must be used to know when to start and stop recording. Then the audio must be recorded in the buffer from where it can be played back.

This is the second section of the patch. Buffer~ and groove~ objects are used to store and playback the recording. This is the case for both the loop and the drum section.

The third section is beat detection. This is to be done by using an algorithm found in the max objects database. The object that was chosen for this project is op.beatitude~ made by Olivier Pasquet in 2005. This algorithm takes in an audio stream as an input and outputs a BPM. This BPM can then be used to set the tempo of the global transport and therefore of the drums.

The fourth section of the patch is the global transport. Global transport is a feature of MAX/MSP that allows the user to set a master tempo that will be followed by the whole patch. This allows for synchronization of different audio files. It is more precise than the metro object which can have some trouble with synchronization due to the start-up of the object.

The fifth section of the patch is the drum section. Drums are read from an audio file. Typically a .wav file. This section is very similar to the audio loop. It will therefore be analyzed in conjunction with the loop section.

The final section is the start/stop buttons. These are simply buttons to start and stop the audio playback. Their bangs are sent via the send and receive objects which trigger a few components to start and stop the playback.

2.1.1 Trigger the recording

The first part of the looper is the foot switch. This is used to start and stop recording. To start and stop recording, you must simply send a bang into the record~ object. This is achieved by analyzing channel 2. This is where a foot switch or trigger of some sort should be connected. This signal is first amplified by a factor of 75. This is simply to make sure the trigger is well received. While

doing tests it was realized that the foot switch that was used didn't always saturate. Therefore this amplification makes sure that the trigger is well received. A meter~ object then convert the MSP signal into a MAX value. After that a number object is used to limit the value to 1. This is then fed into a change object that outputs a number only if there is a change on the input. Finally a select object is used to output a bang only if the input matches 1. This is finally fed into the record object. This section can be seen as a way of converting a signal into a bang that is triggered when the channel saturates.

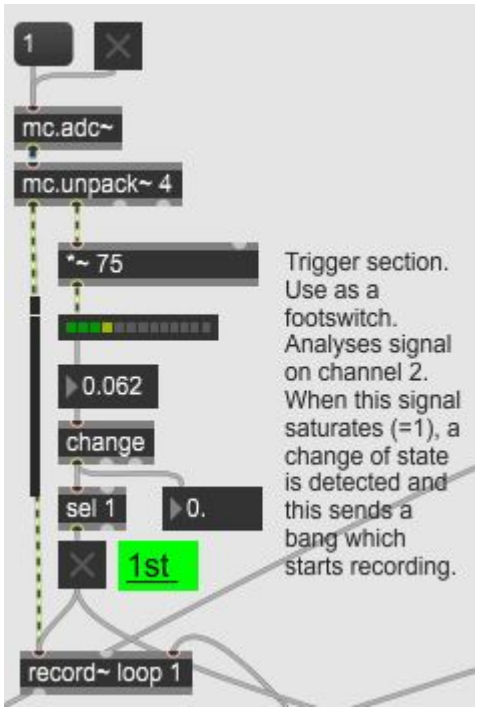

Figure 1: Trigger section of the looper

The next section is the use of the timer object. This is used to determine the total length of the loop. The bang from the foot switch section is fed into a bangbang object. The outlets of this are sent to the timer object. This calculates the total time of the loop which is then fed into the record~ and groove~ objects to tell the recording end point Loop maximum.

2.1.2 Buffer~/Groove~ loop and drum

The following section of the patch is the use of the buffer~ and groove~ objects in conjunction. The setup is the same for both the loop and drum parts. Therefore they will be analyzed together. The basic principle is that a buffer of 30000ms is declared with the buffer~ object. The buffer is then controlled via the groove~ object. To link the two they must simply have the same first argument, in this case "loop" or "drum". In the case of the loop section, the original buffer length is 30000ms. The is then cropped to the length that is determined by the timer object. This cropping must be

done so that the groove~ object can properly determine the tempo and length. Audio files can be opened by the buffer~ object. Before the file is read by the buffer, a new size of 30000ms is assigned to the buffer. This assures that the buffer will be long enough to accommodate the file. Otherwise, if the new file is longer than the previous one, the extra length will not be saved.

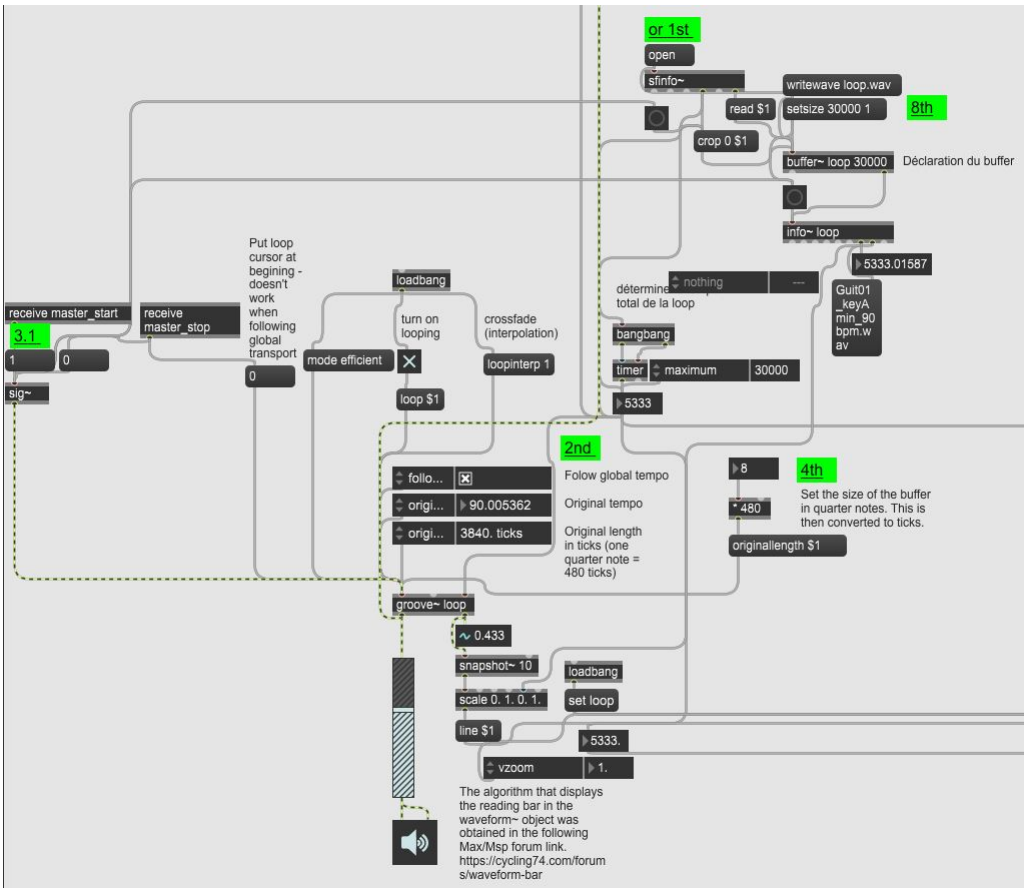

Figure 2: Use of buffer~ and groove~ with all their different components

A big part of the project happens in the groove~ object. The groove~ object allows for playback at different rates. Most notably, it allows the audio file to be played back in sync with the master tempo of the Global Transport. The way this is done is by specifying the length of the played back file in ticks. Ticks are defined as being 480 ticks per beat. Therefore the number of beats of the loop needs to be specified. The groove~ object then deduces the tempo of the loop. With the followglobaltempo attribute selected, it will play back the audio in sync with the global transport. The output of the groove~ object is then sent to a gain~ object and then to a digital to analog converter object.

A waveform viewer is a useful tool that allows for a quick visual analysis of what is happening in the associated buffer. This is achieved with the waveform~ object. There are only three parts to this objects. First, it is told to which buffer~ it is associated. This is achieved by the "set loop"

message. Secondly, the total display length is set. Finally, a scrubber bar is passed through the waveform~ object. This allows for a quick view of where the playback is. This scrubber bar is achieved with the help of the second output of the groove~ object. This is a signal that goes from 0 to 1 and represents where the playback is. The algorithm to create a bar that goes through the waveform~ object was taken from the MAX/MSP forum. A link can be found in the patch.

2.1.3 Beat detection

The beat detection section is used to set the global transport tempo. This way, it is not up to the user to determine the loops tempo. This beat detection is done by using op.beatitude~ object that was designed by Olivier Pasquet. This was found in the max objects database. To use this object, simply feed it the MSP signal from which that is to be analyzed. Turn on the object, set the memory length and finally give the object a minimum and maximum tempo. If this setting is too wide, the result of the analysis won't be right.

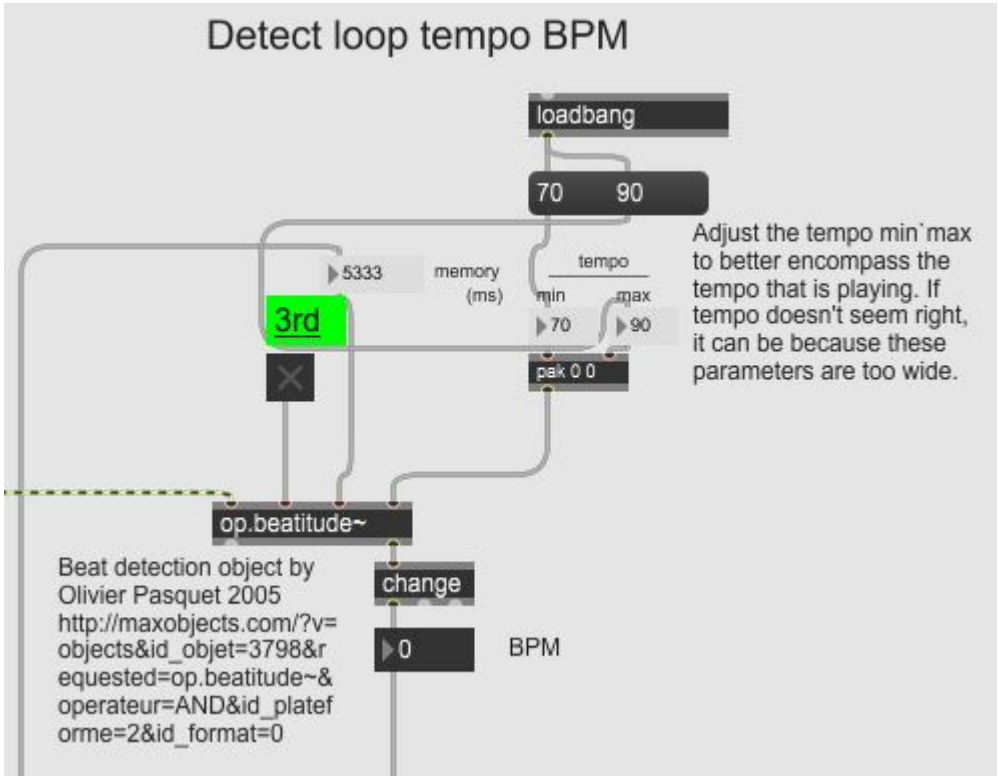

Figure 3: Beat detection

This object does suffer from reliability issues. As mentioned, if the min/max tempo setting is too wide, the object will return a bad tempo. Even if these settings encompass the tempo correctly, the tempo might still be off by a few BPM's. In the case of this patch, it is not a big problem since it will only change the playback rate and not the audio analysis.Furthermore, even if the

object is turned off, it can sometimes still change the tempo. One big issue is that the object will dynamically change the tempo. This means that it might determine multiple tempos for a single loop. This can be an issue if the loop that is played isn't perfectly on beat. For example, if it was recorded live, there might be some slight variations in tempo played by the musician. This will cause the object to change its output value. This has to be remedied by the user who has to analyze the BPM and determine the one that is right.

2.1.4 Transport bar

The transport bar is simply used as a global metronome. The tempo that is determined in the beat detection section is fed into the tempo setting of the transport object. The groove~ objects sync to the global transport. That is how they manage to stay in sync.

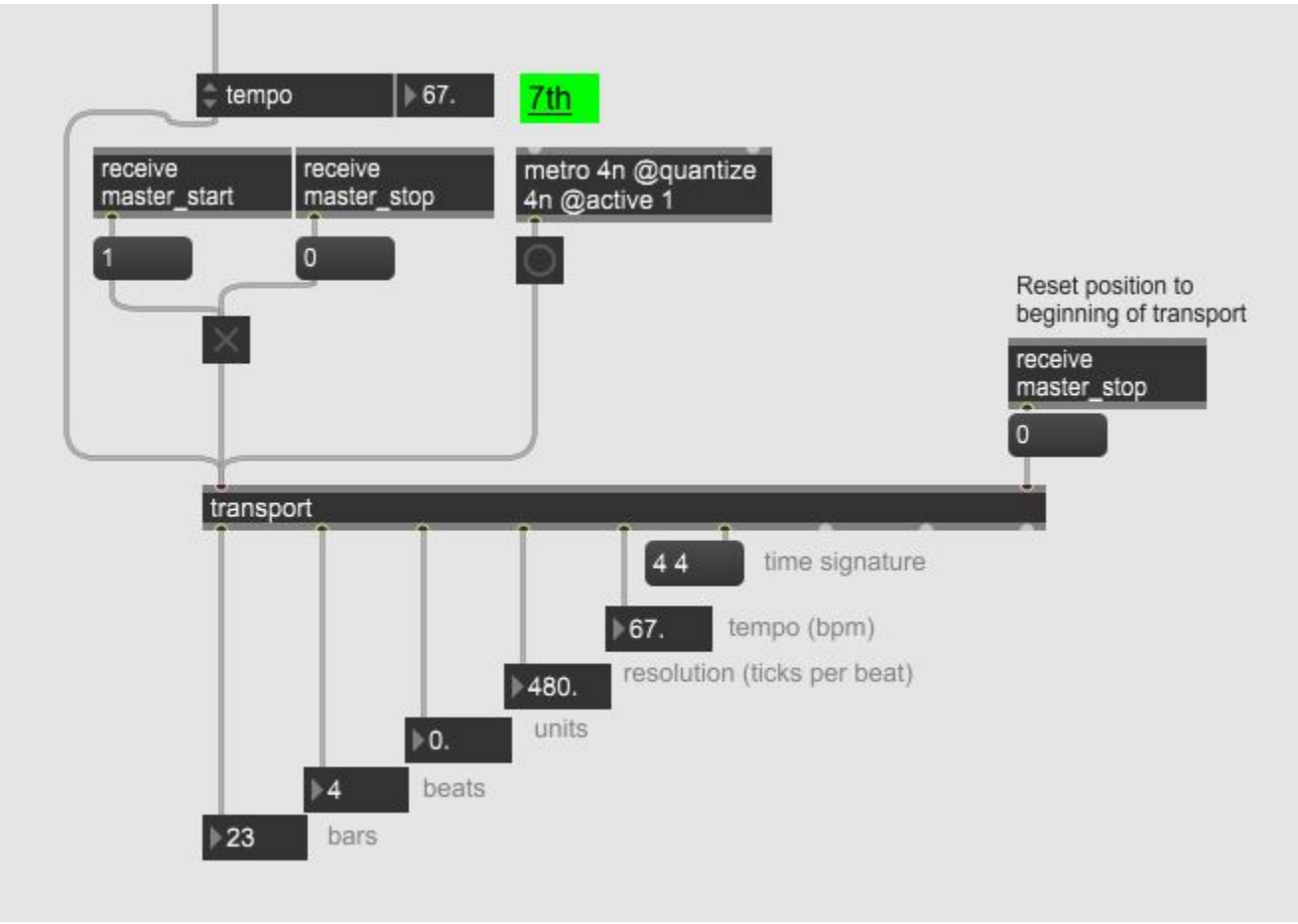

Figure 4: Transport object use to set the tempo of Global Transport

The use of the transport bar also allows for the user to change the tempo of the playback by

changing the value of the tempo setting. Finally, to start the buffers at the same point, they are always started at the beginning by sending a "0" message in the right inlet of the transport bar. Thus setting the position in ticks to 0.

2.1.5 Master start/stop

The last section of the patch is the master start/stop buttons. The start button is used to start the global transport. It also sets the playback speed of the groove object to 1. This means that the file is read at normal speed when it isn't synced to the global transport.

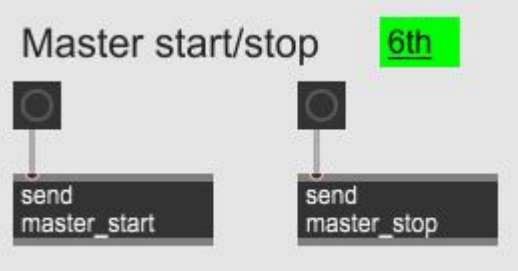

Figure 5: Master Start/Stop

The stop button is used to set to same parameters as the start button. It sets the playback speed of the loops to 0. Essentially pausing them. The also stop the global transport. It also does one more action than the start button. It resets the position of transport to the beginning. This way, when the start button is pressed, it will start the loops from the same point, being the beginning.

3 Issues

There are a few things that proved to be challenging when designing this patch. The first challenge was the analysis of an MSP stream to be used as a trigger. MSP signals aren't designed to be used as triggers and therefore a sort of work around had to be used to properly change the state of the recording object.

The following problem that was encountered was beat or tempo detection. This issue was solved by using an object that was preexisting. This simplified the problem a lot. A lot of research had to be done in order to find an object that worked properly and could be executed in MAX 8. There are a lot of objects out there that seem to solve this problem but unfortunately aren't up to date for MAX 8. Generally, they are 32 bit versions and therefore don't work on a 64 bit computer. The object that was found works under the right conditions but can be tricky to use and it does have some reliability issues as previously explained.

Finally, the biggest issue was that of synchronizing the audio loop and drums. The first iteration of the loop used MIDI drums. This proved very unreliable to synchronize with the audio loop. A bang would be sent to both MIDI and audio loop to start but the MIDI drums would start after the audio loop. Multiple ways were tried to sync the two but nothing worked. A second iteration

used individual samples of kick, hi-hat and snare but this iteration suffered from the same issues as did the MIDI iteration. Finally, the solution that was implemented was to use a prerecorded audio file of the drums. This way the buffer~ object could be used to synchronize to global transport. Furthermore, this also allows for the tempo to be changed at will.

4 Shortcomings

This patch has a few shortcomings. The biggest of them is the fact that the drums need to be prerecorded into an audio file so that they can be fed into the buffer~ object. When first starting this patch, the goal was be be able to play the drums either from individual samples or in MIDI format. For example, the user could chose the drum pattern that was to be played. There is a way to remedy this. It will be discussed in the following section.

The other shortcomings are due to the nature of live recording a loop by a musician. First of all, the pedal that is used isn't necessarily 100% accurate. For example, the pedal I used is a sustain pedal for electric piano. The instant the recording is triggered in the patch might be slightly late compared to the instant my foot touched the pedal. Therefore missing part of the attack of the first beat and messing up the tempo that will be deduced. The second issue is with the nature of a live musician. What is played won't be exactly on beat unless the musician is truly great. These slight errors in tempo will be noticeable on playback especially when compared to the drum at slow tempos. A way to contravene this issue will be discussed in the following section.

5 Future development

As mentioned above, a future development that would be greatly appreciated is the ability for the user to write his own drum tracks/patterns. The could be done by creating a patch that allows the user the create his own patterns. These patterns could then be recorded into an audio file which could then be read from the buffer~ object. Therefore using the patch as it is now and simply adding a functionality.

Another obvious development is to make the patch multi channel. This way the musician can add multiple parts on top of each other, create a much more complex and interesting loop. This can be achieved by using multi channel buffer~ and groove~ objects.

Finally, a way to make the patch more robust for the musician is to implement a metronome count-in that also plays while recording. This will give the musician a reference to keep the beat. This way the tempo will be more consistent during the whole loop. Leading to fewer tempo errors. It can also be envisioned that the trigger that starts the recording wouldn't be from a pedal the user presses but rather this would be done automatically since the patch already knows the tempo. The user would also have to specify a length of the recording so that the patch knows when to stop the recording.

6 Sources

Two parts of the patch were sourced from the internet. The first is the beat detection object. This was designed by Olivier Pasquet in 2005.

http://maxobjects.com/?v=objects&id_objet=3798&requested=op.beatitude~&operateur=AND&id_plateforme=2&id_format=0

The second part of the patch that was sourced from the internet is the scrubbing bar for the waveform object. This was found on the MAX/MSP forums. The author is Tim Lloyd on June 22nd 2009.

https://cycling74.com/forums/waveform-bar