QM-DSP - Bug #1881
keydetection returns an invalid key 25

2019-05-12 09:44 PM - Daniel Schürmann

Status:	New	**Start date:**	2019-05-12
Priority:	Normal	**Due date:**	
Assignee:	Chris Cannam	**% Done:**	0%
Category:		**Estimated time:**	0.00 hour
Target version:			

Description

In the Mixxx project we have discovered a bug that some tracks have are detected as invalid key 25.
This happens due to a rounding issue and is fixed here:
https://github.com/c4dm/qm-dsp/pull/2

History

#1 - 2019-05-15 09:37 AM - Chris Cannam

Thanks for the report!

Do you have an example of an input file for which it produces this output?

The discussion on Github looks entirely plausible, and you are almost certainly in a better position to test this with real-world data at the moment than I am. But I do need to be cautious about updating the core logic of a library and plugin that I did not originally design and that has been producing the same output for over a decade. I'm aware that a "natural" fix to one part of a library can sometimes break the results further up a pipeline (perhaps because something in the plugin code has some magic that "compensates" for it) so this sort of thing is not risk-free.

I will set up some tests for this locally and have a closer look, but any extra info or data you can provide would be handy.

#2 - 2019-05-15 09:37 PM - Daniel Schürmann

- File 1975.wav added

Yes, a 1975.53 Hz sine wave created with Audacity (see attached)
This should be detected as B minor = 24, but returns the invalid value 25.

Playing with these similar sine waves, I have also discovered an other bug.
It is fixed here https://github.com/mixxxdj/mixxx/pull/2112
and a thread save issue fixed here:
https://github.com/mixxxdj/mixxx/pull/2113

Do you know some algorithm insights?

But lets first fix this one here.

#3 - 2019-05-16 10:57 AM - Chris Cannam

- Assignee set to Chris Cannam

Thank you, that's enough to reproduce the problem here as well.

I want to make sure I have enough test scaffold before I bring in any changes, so what I propose to do is to prop up some regression tests for the QM Key Detector plugin (which of course uses this code) and tone-generator unit tests for the key detection logic here. Then I should feel safe applying these fixes. I expect to have time to do this at the start of next week (probably not this week sadly). My purpose here is just to make sure things don't break by accident - despite its faults, this code has done a fair job of detecting keys for users for some time, so we have to make sure any changes are pure improvement.

Looking over your other changes, I do see a few things that we can't pull here - e.g. removing the key strengths, which are used by the key-strength output of the key detector plugin. However you are plainly right about the circular shift on a static array - that is certainly a creative way to do things! I hadn't noticed it before.

Thanks again for having the patience to propose these fixes upstream.

#4 - 2019-05-16 06:18 PM - Daniel Schürmann

Is a GitHub PR the right channel to request merges?

Is it reasonable to issue a syle PR? The major issue is that the code has mixed tab and spaces indent.
It would be great to have that fixed. Do you have a style guide or something?

#5 - 2019-05-23 11:58 AM - Chris Cannam

The way things are set up at the moment, this repository is the authoritative one, and the Github repo is just an automatically-updated mirror of it. So it's not currently possible to merge PRs on Github. However, it can quite easily be switched so things work the other way around. I don't mind doing that for this repo, and managing both code and issues on Github. I will make that switch after committing some tests (this afternoon I expect).

A problem with merges of any kind is copyright: QMUL would like to maintain the ability to sell commercial licences for the code. This happens occasionally, and when it does, much of the licence fee goes to the researchers who did the original published work - not necessarily to anyone who actually implemented it in code - giving a small bonus to researchers who are often not paid all that well. Although this code is old and dusty, it's still "live" as far as licensing is concerned. Therefore, it would be necessary either to assign copyright on contributions to QM, or (probably easier) to include a note that applies a BSD-style licence to any contributions so that they can be included in a GPL library commercially relicensed by QMUL.

About code style - indentation is supposed to be 4 spaces at a time, no tabs, opening brace at end of line except at the start of a function definition. But as you see, some files are almost entirely untouched since being written in a different style. Making them consistent would be a good idea, yes!

#6 - 2019-05-23 07:40 PM - Chris Cannam

I've switched the primary to Github, and have created a branch (chroma-key-tuning-review) with some test material, into which I would like to pull these fixes.

As I understand it, your commits do three things:

1. fix a rounding error in the key detector which treats the chromagram as 1/3 semitone lower than it actually is; this may produce a worse key estimation than it otherwise would, and can occasionally produce an entirely invalid key output
2. fix a rounding error in the chromagram which means that sinusoids are not always placed within the proper bin position?
3. fix thread-safety problems in the key estimator

From initial tests I concur with all of these problems, but there are interesting compatibility issues at play, especially with the second. I would really like to take a look at the second issue first, but since they affect separate code files, I could pull the first two at once.

Would it be possible to submit a PR on Github with both key-detector and chromagram rounding fixes (but not thread-safety fixes yet)?

On the subject of style fixes - I added a note in the new CONTRIBUTING.md about style. I would like to get any existing functional fixes sorted out first, but I can easily go ahead and tidy these things up after that.

#7 - 2019-05-23 08:53 PM - Daniel Schürmann

I don't mind about the license, you can license the code how you like. It is enough for me to see my name in the commit message.

To 1:
The original implementation is centered at a full semitone. The additional initial shift is a trick to avoid dealing with the flat C being at index 35. It is shifted to 0 and compensated by the dropped ceil() call.
This way we have all three C bins at 0/1/2 and the division by int 3 has the same result of 0.

Though I am a bit doubtful if the side bins are valid at all. They are IMHO created either by useless sweeps, off pitch notes or overlapping neighbor notes, which all waters the result.

To 2:
Here I am unsure from the theory we need to place the hamming windows for the kernel in the middle of the fft window. Like tat *db* (db is my ascii art for both halfes of the Hamming window. My patch does this from b___d which should be wrong because we have two rectangulars flanks in the data. I could imagine that the original author wants to make use of all samples in the window like this dTTTTb. I am not sure if the constant Q theory is still valid than. Are you able to verify this?

To 3:
Yes.

OK, I can update the PR as requested. I hope I will find time soon.

#8 - 2019-05-24 10:07 AM - Chris Cannam

About the shifted window: I don't have a definitive answer, but the usual purpose of a half-frame shift like this (in the input to an FFT) is to correct the phase of the output.

A short-time Fourier transform treats its input as part of an infinite signal that repeats indefinitely in both directions. So if your short-time input looks like (using the same ascii art) __db__, then as far as the STFT is concerned the signal is actually an infinite one like ...__db____db____db__... etc. If we shift the input by half of its length like b____d, then the STFT sees an infinite signal consisting of ...b____db____db____d... - which is the same signal shifted by a half-frame, there aren't any discontinuities in it because the edges are wrapped around seamlessly at each repeat. It only differs in placing the zero phase at the peak of the window.

This shift is almost always applied to a windowed signal frame used as input to any method that does phase-based manipulation of the frequency-domain signal, such as a phase vocoder. I'm less clear about the rationale here - it will shift the phases of the frequency-domain kernel bins but why that should be necessary, I'm not sure - other CQ implementations that I'm more familiar with do not do this. I'll try to take a closer look.

(Of course, if one tried to do this shift but got it wrong, that would be a big problem - e.g. an off-by-one error in the distance to be shifted would lead to a zero appearing right at the peak of the window, which would obviously lead to an extremely noisy result. Probably ought to check for that as well...)

#9 - 2019-05-24 09:15 PM - Daniel Schürmann

I have just updated the PR https://github.com/mixxxdj/mixxx/pull/2113
I am happy to receive your comments.

#10 - 2019-05-29 12:16 PM - Chris Cannam

I presume the URL in your last comment was intended to be https://github.com/c4dm/qm-dsp/pull/2

I've pulled this to the chroma-key-tuning-review branch - thank you! I've read through the changes here and am happy with them.

With regard to the constant-q / chromagram, I took a look at the constant-q output with and without the phase shift to zero-align the kernel window shape, and whatever the reason for it, it seems that this shift is necessary in this implementation - the output (e.g. from the constant-q spectrogram QM Vamp plugin) is quite wrong for simple sine tones without it. I would like to take a look at the other possible problems in the chromagram code that you identified - do you have a branch you can point me to that has these fixes in it?

Thanks!

#11 - 2019-05-29 02:08 PM - Chris Cannam

I decided I could probably take some of these fixes directly from the diffs for your Mixxx branch, so I have now applied the fixes from your commits 957aed58 Use double precision for FS and b9c6bde3 Extend the number of Q bins always to a full octave. Both of these look fine in principle, but the second one changes the output of one of the other QM Vamp plugins (the tonal-change function plugin), so I'm going to review that as well and see if there's any problem there.

#12 - 2019-05-29 03:57 PM - Chris Cannam

Ah, I see what the problem is with the constant-q phase shift! There is indeed a very significant error here, it just isn't what it seemed at first.

The Chromagram class has two process methods: one for frequency-domain input and one for time-domain input.

The reason the window shift is necessary, and the Chromagram Vamp plugin produces the wrong output without it, is that the plugin itself uses the frequency-domain input version of the process method, and it feeds phase-corrected frequency-domain data to it. The phase shift in the CQ kernels is therefore necessary to match the input.

Unfortunately, the **time-domain** version of the process method fails to carry out the same phase shift, and so the kernel doesn't match. This explains almost the whole error you are seeing. I've committed a fix in the chroma-key-tuning-review branch on Github.

There is still a small drift for input frequencies close to the bottom of the spanned frequency range, which I'm investigating.

#13 - 2019-05-30 04:28 PM - Chris Cannam

I've fixed the drift for lower frequencies as well - it turned out to be a bug I introduced myself a long time ago while fixing something else. I was more lax about automated testing back then.

This seems fairly sound now - are there any further problems you've noticed? (I mean wrong output, in particular, rather than code quality?)

Many thanks for your input with this.

I've merged back to master and am about to make a branch for code style fixes.

#14 - 2019-06-03 11:29 PM - Daniel Schürmann

Great, almost all noise is gone now from the chroma. Thank you very much.

One interesting part is that the hop size is currently ignored.
https://github.com/c4dm/qm-dsp/blob/884fbb26af0190ed108097a8cc74fb62646d5e36/dsp/keydetection/GetKeyMode.cpp#L86

This speeds up the key detection by the factor 8. But is this appropriated? If I re-enable the hop size, I get significant other max bins for the skipped 7 detection results. I think this is caused by the small Hanning window.
Later we add a moving average filter to the bins. So it might be reasonable to take all 8 measured values into account and not only a random sample.
Is there a way to analyses the whole audio frame in one step without braking the Q value required for the constant Q analysis?

One other issue in the Mixxx context is that the results are not reliable because even a non musical noise track is detected with a key. We use the key info to find a compatible track and if such a track is considered as compatible is sounds not as good as desired. For this we need a feature that returns an invalid key for such tracks. The same is true for non western tracks or tracks with a non typical scale like Harmonic Moll. In this case it is much better to not return a key instead of returning the most likely key. The later makes the analyses of all tracks less trust worthy.

For this, I have tried a different approach see: https://github.com/mixxxdj/mixxx/pull/2135 What do you think about it? Do you have interest to make this feature mature.

#15 - 2019-06-04 04:22 PM - Chris Cannam

I was thinking the same thing about the hop size yesterday. I agree with you - it's just throwing information away, or taking a random sample as you (more charitably) put it.

Maybe there was a deliberate tradeoff between speed and measured accuracy that seemed worthwhile at the time this algorithm was written - it certainly makes the method very fast.

It is possible to make a CQ algorithm that requires less overlap in its input frames, by bundling multiple offset kernel atoms into a single frame at higher frequencies. But this one doesn't do that. An example of a library that does do so is https://code.soundsoftware.ac.uk/projects/constant-q-cpp - though that library's interface handles its own framing anyway, so its input is just non-overlapped time-domain data.

For the qm-dsp/qm-vamp-plugins code I will probably try making this an option - perhaps some user might value the speed of the method more than precision.

With regard to your other proposed changes, the principle seems sensible and interesting, but I think this is really a different algorithm and it probably isn't on topic for qm-dsp - a library whose defining characteristic of course is that it contains algorithms from QM.

I wondered if it might be worth making a separate dedicated key-detector library for experiments like this. I created this repo as a possible idea: https://github.com/cannam/keydetector. It contains the qm-dsp key detector code (from the current `codestyle-and-tidy` branch) as well as your code from the above-linked Mixxx PR (I applied the same codestyle and tidy updates to that as well, though I'm afraid I haven't had time to test the results yet!) There are tests in qm-dsp that could be brought across as well. Just a possibility - see what you think.

Other possibilities might include building the same code against a different chromagram implementation - there are several options that have different tradeoffs in terms of e.g. how they avoid interference between fundamentals and higher partials. Some examples are the NNLS Chroma chromagram (https://code.soundsoftware.ac.uk/projects/nnls-chroma/), Tipic (https://code.soundsoftware.ac.uk/projects/tipic), or the other implementation I linked already (https://code.soundsoftware.ac.uk/projects/constant-q-cpp).

Of course, much may depend on the needs of the Mixxx codebase.

One other remark is that I routinely submit plugins as entries to the annual MIREX evaluation (see https://thebreakfastpost.com/2019/03/01/mirex-2018-submissions/ and https://www.music-ir.org/mirex/wiki/MIREX_HOME). I will probably submit both old ("broken") and new ("fixed") versions of the qm-dsp key detector as separate entries this year in order to verify that the fixed version does actually perform better. If there are other interesting algorithms here, perhaps they could be submitted as well.

Downloads

1975.wav	2.52 MB	2019-05-15	Daniel Schürmann