Audio Engineering Society

Convention Paper

Presented at the 137th Convention
2014 October 9–12 Los Angeles, USA

A Comparison of Real-Time Pitch Detection Algorithms in SuperCollider

Elliot Kermit-Canfield[1]

[1] *Center for Computer Research in Music and Acoustics, Stanford University, Stanford, CA, USA*

Correspondence should be addressed to Elliot Kermit-Canfield (`kermit@ccrma.stanford.edu`)

ABSTRACT

Three readily-available pitch detection algorithms implemented as unit generators in the SuperCollider programming language are evaluated and compared with regard to their accuracy and latency for a variety of test signals consisting of both harmonic and non-harmonic content. Suggestions are made for the type of signal on which each algorithm performs well.

1. INTRODUCTION

Pitch detection is not a new area of research and there are many algorithms and techniques which accomplish this task. Pitch detection is the process of identifying the fundamental frequency, $F0$, of a signal. Although the goal of these algorithms is *pitch detection*, they really perform *pitch estimation*, and throughout this paper, these terms will be used interchangeably. This paper does not seek to submit a new technique for pitch detection, but rather investigates three pitch detection algorithm implementations in the SuperCollider audio programming environment [1].

SuperCollider is an open source audio programming environment which is well suited for many computer/audio tasks ranging from composition to performance to analysis. Since SuperCollider supports real-time audio, pitch detection algorithms must meet several criteria—they must detect pitch with accuracy and stability but also run with low latency and low computational load. These features are all important for a robust and reliable pitch detector.

In the following section, the three pitch detection algorithms are explained. Following that, section 3 introduces the test signals and methods for evaluation. Sections 4 and 5 discuss and evaluate the results of the evaluations, and the final section provides some concluding remarks and suggestions for further research.

2. PITCH ESTIMATION ALGORITHMS

There are three readily available pitch estimation implementations in SuperCollider. Each is implemented as a unit generator (UGen) that outputs two values at control rate, one for if the algorithm has detected pitch in that block (*hasFreq*) and a value for the detected pitch (*freq*).[1] One of these UGens, the so-called Pitch UGen, comes with the standard SuperCollider download while the other two, Tartini and Qitch, come packaged with the SC3 plugins [2].[2]

2.1. Pitch Autocorrelation Algorithm

The Pitch UGen is an implementation of perhaps the most commonly used pitch detection scheme. It is an autocorrelation function (ACF) based pitch detector. The ACF finds periodicity of the input signal by taking a series of correlations between the input signal and a time delayed copy of itself at increasing lags l. It is defined as

$$r(l) = \sum_{n=0}^{N-1} x(n)x(n-l). \quad (1)$$

At lag zero, the autocorrelation is at its maximum and typically decreases as the lag increases. Since pitched signals are periodic, they exhibit a special property where their autocorrelations are also periodic. This causes the autocorrelation to be at a minimum when the lag reaches half the period of the fundamental frequency, at which point it should rise again. We should observe maxima that peak periodically at the fundamental frequency.

The implementation of the ACF in the Pitch UGen is well designed.[3] It accepts a *minimum frequency* and *maximum frequency* which constrain acceptable lag values. Operating at the *execution frequency*, it does a rough pitch estimation by taking the autocorrelation of the input at a resolution of *maxBinsPerOctave* and finds the first peak after the peak at lag zero that is higher in amplitude than the *peakThreshold* multiplied by the value of the autocorrelation at lag zero.[4] Once it finds that peak, it does a finer resolution search before performing parabolic interpolation to achieve its final pitch estimation. Additionally, there are several features and safety measures built into Pitch. Pitch allows for downsampling before performing the pitch estimation to reduce computational load, and median smoothing, which provides more stability and robustness at the output, although it adds some latency. One can also specify a *clarity* flag, which causes Pitch to output the height of the autocorrelation peak normalized by the height of the peak at zero lag instead of simply a boolean value for whether or not a pitch was detected.

2.2. Tartini Frequency Domain Algorithm

The Tartini UGen is an implementation of Philip McLeod's Tartini algorithm which employs an FFT-based normalized squared difference function (SDF) method [3, 4]. The SDF can be defined as

$$d(l) = \sum_{n=0}^{N-1} (x(n) - x(n-l))^2. \quad (2)$$

Multiplying this out, it can be seen that there is an embedded ACF

$$d(l) = \sum_{n=0}^{N-1} (x(n)^2 + x(n-l)^2 - 2x(n)x(n-l). \quad (3)$$

By changing the length of the window as a function of l

$$d'(l) = \sum_{n=0}^{N-1-l} (x(n) - x(n-l))^2, \quad (4)$$

there is a tapering effect as l increases caused by the decreasing size of the window. By defining

$$m'(l) = \sum_{n=0}^{N-1-l} (x(n)^2 - x(n-l)^2), \quad (5)$$

and by rewriting the ACF to allow the window to change as a function of l

$$r'(l) = \sum_{n=0}^{N-1-l} x(n)x(n-l), \quad (6)$$

[1] SuperCollider allows unit generators to output values at audio rate (e.g. 44,100 Hz) or at a slower control rate. SuperCollider defaults to a block size of 64, yielding a control rate of about 689 Hz.

[2] The SC3 plugins contain commonly-used extensions to SuperCollider not deemed essential or finished enough to be packaged with the main source.

[3] The input arguments and their default values for the Pitch, Tartini, and Qitch UGens can be found in their help files.

[4] The execution frequency must be between the minimum and maximum frequencies.

we can write the SDF as

$$d'(l) = m'(l) - 2r'(l). \tag{7}$$

In McLeod's Tartini algorithm, the squared difference function is normalized and defined as

$$n'(l) = 1 - \frac{m'(l) - 2r'(l)}{m'(l)}. \tag{8}$$

This bounds the output to $[-1, 1]$, where 1 is perfect correlation, -1 is perfect negative correlation, and 0 is no correlation.

For efficiency, this is then implemented in the frequency domain. In SuperCollider, the user has control over the size of the FFT and the FFT window hop size. Additionally, the user can set parameters that constrain how many peaks in the autocorrelation are considered as well as a threshold for how high peaks must be in relation to the peak at lag zero. Like the Pitch UGen, Tartini then performs peak picking on local maxima and performs parabolic interpolation on the most likely peak. Tartini can also be configured to provide a clarity factor loosely corresponding to confidence of the pitch estimation.

2.3. Qitch Constant Q Transform based Algorithm

Qitch is a constant Q transform based pitch detection algorithm created by Judith Brown and Miller Puckette [5–7]. This pitch estimation algorithm relies on a perceptual model implemented as a constant Q filterbank. The center frequencies are geometrically spaced in quarter steps according to

$$\omega_{k_c} = (2^{\frac{1}{24}})^{k_c} \omega_{min}, \tag{9}$$

where k_c is the k^{th} filter and ω_{min} is the lowest center frequency. The bandwidth is defined to be $\omega_{k_c+1} - \omega_{k_c}$, which keeps Q constant.

In the instantiation of the Qitch UGen in Supercollider, one must link to a *.wav* file containing the kernels to the FFT. This makes using Qitch challenging, as one must load the kernel file into an audio buffer and then pass the buffer to whatever function is utilizing Qitch. Qitch allows the user to influence the algorithm's performance by supplying a buffer containing an amplitude weighting contour for the kernels, minimum and maximum detectable frequencies, and a flag for refining the pitch estimation based on instantaneous phase estimation.

3. METHODS

The three pitch detectors were tested with their default settings in response to a variety of test signals. In this paper, the test signals include chromatic scales synthesized with pure tones (single sinusoid), recorded guitar scales, and linear, sinusoidal chirps. These signals range in terms of the complexity of their spectra over time, noise floor, pitch stability, and pitch range. More signals—including synthesized Karplus-Strong plucked strings and FM bells, as well as recorded piano and voice stimuli—were tested, but due to the brevity of this paper, they are not included in this analysis.[5]

Each pitch detector is evaluated with each signal to see how it performs against the other detectors. We are interested in how the performance of the SuperCollider UGens compare to each other in terms of their accuracy, latency, robustness, and computational efficiency. Some algorithms work well on specific types of signals but do poorly on others.

Because of SuperCollider's language/server configuration there is a substantial amount of overhead to study the pitch detectors. Reusable SuperCollider code was written that encapsulates the pitch detectors in SynthDefs, which in SuperCollider are server constructs primarily consisting of UGen graphs. These SynthDefs run in parallel and contain open sound control (OSC) triggers, which output the values of *freq* and *hasFreq* for their pitch detector once every block. In the language, OSC responders listen for these events and write them to *.csv* files, see Figure 1.

4. RESULTS

4.1. Chromatic Sinusoidal Scale

This is the simplest test case. The stimulus consists of a chromatic progression of two-second long pure sine tones ranging from C_3 to C_4 multiplied by Hann window amplitude envelopes, see Figure 2. This signal tests how the pitch detectors handle a harmonically deficient signal.

[5] All audio files, Matlab and SuperCollider code, and additional analytical graphics can be downloaded at `http://ccrma.stanford.edu/~kermit/scpitch`.

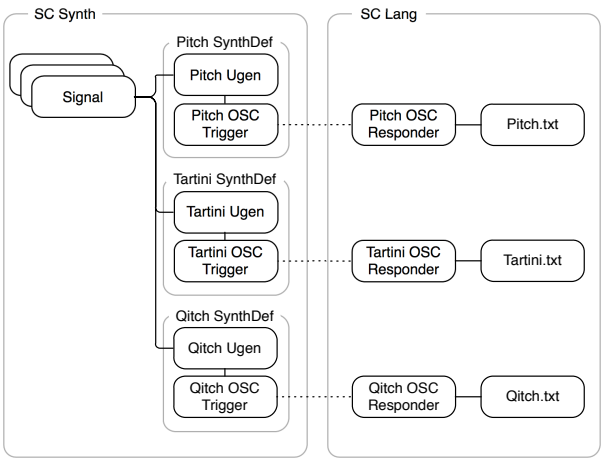

Fig. 1: SuperCollider Language and Server Routing

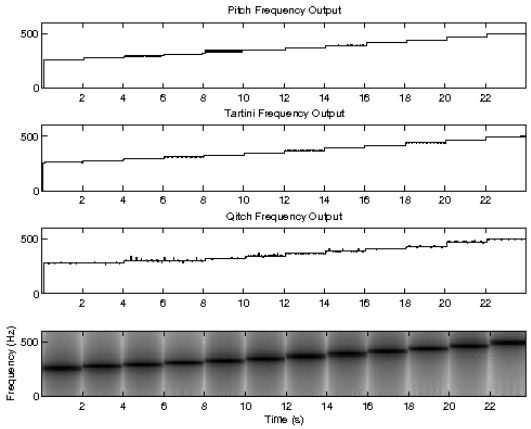

Fig. 2: Chromatic Sinusoidal Scale from C_3 to C_4

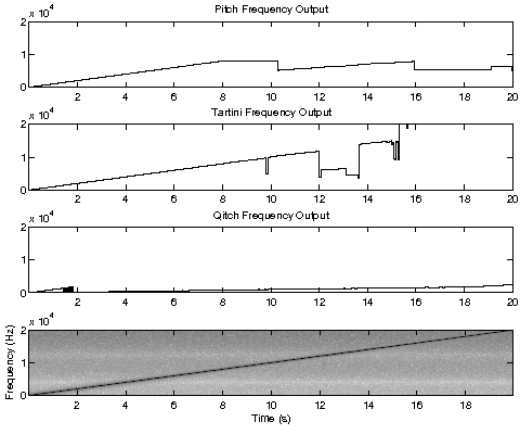

Fig. 3: Sinusoidal Chirp from 20 Hz to 20 kHz

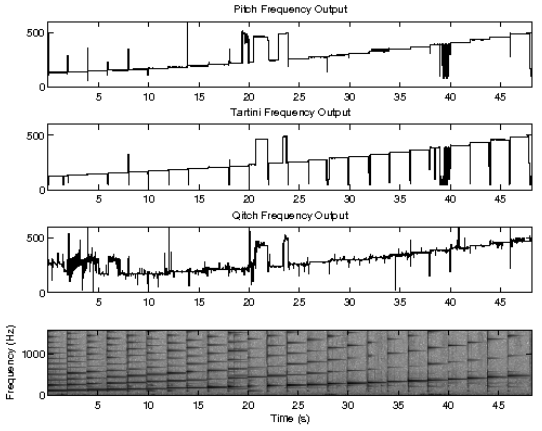

Fig. 4: Chromatic Guitar Scale from C_3 to C_4

4.2. Sinusoidal Chirp

This test stimulus consisted of a full spectrum (20 Hz to 20 kHz) linear chirp, see Figure 3. Unlike the scale example, this allows us to look at how the pitch detectors do when confronted with a signal that is constantly changing. Additionally, since we are testing the detectors with their default settings, the frequency of the chirp exceeds the range of detectable frequencies for all three detectors. This allows us to see what happens when a pitched signal is present that does not fall cleanly into the detectable range.

4.3. Chromatic Guitar Scale

The last stimulus presented here is a recorded chromatic guitar scale from C_3 to C_4, see Figure 4. The guitar's spectrum contains a large number of harmonic overtones that decay quicker as they rise in frequency. This allows us to investigate the detector's response to a harmonically rich spectrum that changes over time. Unlike the synthesized examples, this stimulus has a higher noise floor due to

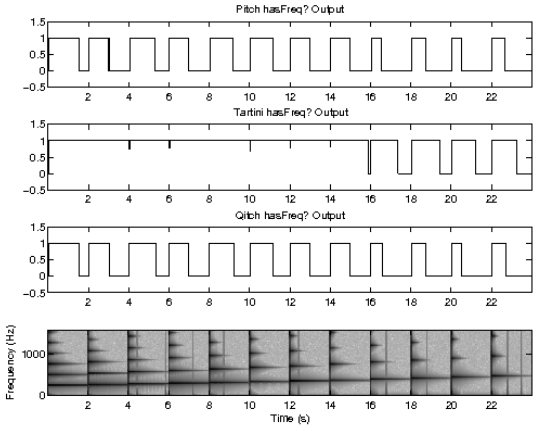

Fig. 5: Output of *hasFreq* for a Synthesized Karplus-Strong Plucked String Chromatic Scale from C_3 to C_4

the recording process.[6] Additionally, because it is recorded and not modeled, there is significant variance from note to note.

Although we acquired and analyzed the *hasFreq* output for the three stimuli presented in this paper, plots of their *hasFreq* outputs are not useful from a visual perspective. Figure 5 shows the output of *hasFreq* for the three detectors in response to a synthesized Karplus-Strong plucked string chromatic scale. A value of 1 means the pitch detector has estimated $F0$ while a value of 0 means the detector is unable to find $F0$ with enough confidence.

5. DISCUSSION

Pitch, Tartini, and Qitch all performed well on the sinusoidal scale and less well on the other two stimuli (Figure 2). Pitch and Tartini can be extremely accurate in their estimates while Qitch is frequently inaccurate and has problems making stable estimates. By analyzing the data output from *freq* and *hasFreq*, it is clear that Tartini has the lowest amount of latency and Pitch has the largest amount of latency. On average, Tartini makes its pitch estimate 1.4 times faster than Qitch and 1.6 times faster than Pitch. Tartini takes about 0.05 seconds to make an estimate with good data compared to 0.07 for Qitch and 0.08 for Pitch.

In response to the sinusoidal chirp (Figure 3), Pitch and Tartini performed reasonably well, but Qitch failed to estimate the correct pitch. Both Pitch and Tartini have issues estimating pitch once the chirp exceeds the upper bound of their *maximum frequency* constraint. It is interesting to note that both of these algorithms make the same type of mistake, which is an octave error. Due to the periodicity of the ACF and the SDF, the algorithms pick an incorrect local maxima causing both algorithms to find a pitch, albeit inaccurate, *modulo 2*.

In the guitar example (Figure 4), it is interesting to see that Pitch and Tartini do a good job for most of the sequence. Naturally, in the time between each pair of notes, the noise floor and decaying previous note cause all three algorithms to have issues. In Figure 5, note how the *hasFreq* output drops when the higher harmonics fade out.[7] Pitch handles this issue well by holding on to the previously detected note until it has high enough confidence to make its next estimation. Tartini, on the other hand, outputs glitchy data between the notes. Qitch's data for the entire sequence is noisier, most likely due to spectral leakage in the calculation of the constant Q transformation. Qitch especially seems to have issues finding the lowest pitches, which is probably due to the low resolution of the constant Q filterbank at low frequencies. It is also interesting to note that at around twenty seconds, the fundamental pitch decays quicker than the first harmonic, causing all three algorithms to estimate an incorrect frequency.

On a modern laptop computer, the CPU load for all three pitch detectors is negligible.[8] Running each pitch detection UGen independently, the average CPU load never rises above 0.5% and the peak CPU spikes are never higher than 1.5%. Naturally, CPU usage could be more of an issue when doing other simultaneous processing or on a less powerful computer.

From a use standpoint, both Pitch and Tartini make the average user's setup easier than Qitch. Both

[6] The recordings were made in a quiet recording studio with high quality equipment.

[7] This figure is from a synthesized plucked string rather than the recorded guitar.

[8] 2013 MacBook Pro running OSX with a 2.8 GHz Intel i7 processor and 8 GB RAM.

of them can be used with their default settings to yield satisfactory results. Qitch requires the extra overhead of loading the FFT kernels manually to make the algorithm work. Beyond that, Pitch allows for the largest amount of customization. The median smoothing feature and ability to downsample to reduce CPU load are great benefits to using Pitch. At the same time, Tartini supplies the user with a simplified list of controls compared to the other UGens. All three UGens allow power users to influence the inner works of the pitch detection algorithms by changing parameters such as FFT size.

6. CONCLUSION

As it turns out, both Pitch and Tartini are robust pitch estimators that work on a wide variety of signals. We would not hesitate to recommend both of them to SuperCollider users. A user might need to try both of them before choosing which one is best for a specific job. Some of the advanced features of Pitch, such as median smoothing, might make it more attractive from a power-user perspective, although Tartini has less latency in making its pitch estimations. Qitch is not the most attractive pitch estimater due to complexity in its use and lack of accuracy.

Future experiments involve increasing the vocabulary of test signals, tweaking the algorithms, and implementing other pitch estimation schemes in SuperCollider.[9] Other pitch detection algorithms, such as spectral peak picking and the YIN algorithm have proven to work well in real-time audio contexts [9–11]. Additionally, chord recognition and other multiple $F0$ recognition tasks are on the cusp of pitch estimation research [12, 13].

7. REFERENCES

[1] www.supercollider.sourceforge.net

[2] www.sc3-plugins.sourceforge.net

[3] www.miracle.otago.ac.nz/tartini

[4] P. McLeod and G. Wyvill, *A Smarter Way to Find Pitch* (2005), ICMC Proceedings: 138–141.

[5] J. Brown and M. Puckette, *An Efficient Algorithm for the Calculation of a Constant Q Transform* (1992), Journal of the Acoustical Society of America, 92(5): 2698–701.

[6] J. Brown, *Musical Fundamental Frequency Tracking Using a Pattern Recognition Method* (1992), Journal of the Acoustical Society of America, 92(3): 1394–402.

[7] J. Brown and M. Puckette, *A High-Resolution Fundamental Frequency Determination Based on Phase Changes of the Fourier Transform* (1993), Journal of the Acoustical Society of America, 94(2): 662–7.

[8] A. Knesebeck and U. Zölzer, *Comparison of Pitch Trackers for Real-Time Guitar Effects* (2010), Proceedings of the 13th International Conference on Digital Audio Effects (DAFx-10), Graz, Austria: 6–10

[9] J. Wise, J. Caprio, and T. Parks, *Maximum Likelihood Pitch Estimation* (1976), IEEE Transactions on Acoustics, Speech, Signal Processing, 24: 418-23.

[10] B. Doval and X. Rodet, *Fundamental Frequency Estimation Using a New Harmonic Matching Method* (1991), Proceedings of the 1991 International Computer Music Conference, Montral: 555-8.

[11] A. Cheveigné and H. Kawahara, *Yin, Fundamental Frequency Estimator for Speech and Music* (2002), Journal of the Acoustical Society of America, 111(4): 1917–30.

[12] A. Klapuri, *A Perceptually Motivated Multiple-F0 Estimation Method* (2005), Proceedings of the IEEE Workshop on Applications of Signal Processing to Audio and Acoustics, New Paltz, NY: 1–4.

[13] www.music-ir.org/mirex

[9] See also Knesebeck and Zölzer [8].