Minimising latency of pitch detection algorithms for live vocals on low-cost hardware

Matthew Firth
Email: u1168395@unimail.hud.ac.uk

Abstract
A pitch estimation device was proposed for live vocals to output appropriate pitch data through the musical instrument digital interface (MIDI). The intention was to ideally achieve unnoticeable latency while maintaining estimation accuracy. The projected target platform was low-cost, standalone hardware based around a microcontroller such as the Microchip PIC series. This study investigated, optimised and compared the performance of suitable algorithms for this application.

Performance was determined by two key factors: accuracy and latency. Many papers have been published over the past six decades assessing and comparing the accuracy of pitch detection algorithms on various signals, including vocals. However, very little information is available concerning the latency of pitch detection algorithms and methods with which this can be minimised. Real-time audio introduces a further latency challenge that is sparsely studied, minimising the length of sampled audio required by the algorithms in order to reduce overall total latency.

Thorough testing was undertaken in order to determine the best-performing algorithm and optimal parameter combination. Software modifications were implemented to facilitate accurate, repeatable, automated testing in order to build a comprehensive set of results encompassing a wide range of test conditions.

The results revealed that the infinite-peak-clipping autocorrelation function (IACF) performed better than the other autocorrelation functions tested and also identified ideal parameter values or value ranges to provide the optimal latency/accuracy balance.

Although the results were encouraging, testing highlighted some fundamental issues with vocal pitch detection. Potential solutions are proposed for further development.

Keywords: Pitch detection; pitch estimation; autocorrelation; algorithm; latency; correlation; signal processing; audio processing; microcontroller; embedded system.

Introduction

This work aimed to identify the most suitable pitch detection algorithm to implement in a standalone vocal pitch detection system for use with a live (as opposed to prerecorded) audio signal. The projected target platform was low-cost, standalone hardware based around a microcontroller (such as the Microchip PIC series), presenting challenges in terms of the limited processing power and low memory typically available on these platforms.

As a creative tool, this device would allow a performer to simply 'hum' a melody to generate the appropriate MIDI note messages, enabling even novice composers to create music in digital audio workstation (DAW) software using only their voice. With a sufficiently low latency, this could also find application in live music, allowing musicians to drive synthesisers by voice. Consequently, such a device must undertake real-time analysis of incoming audio, with minimal (ideally, completely imperceptible) delay between note onset and MIDI transmission.

The ideal algorithm should accurately determine pitch without noticeable latency. However, it was predicted that accuracy and speed of analysis would be conflicting factors and therefore, the aim was to establish an optimal balance between these two factors. Initial research studied various algorithms to determine which would be most likely to achieve the required result acquisition speed while maintaining satisfactory accuracy. Algorithm optimisations were then developed to minimise latency and maximise accuracy further.

Background research

The practice of pitch detection is to extract the fundamental frequency (F_0) from a signal that may be crowded with additional frequencies, and determining its pitch (Gerhard, 2003). The aim of the research phase was to build an understanding of the types of algorithm used for F_0 estimation to form an evaluation of their suitability and the potential challenges that may be faced.

Many studies have already been conducted in the area of pitch detection, and extensive information is readily available. However, these studies have largely focused on the accuracy of pitch estimation, with very little attention given to latency and methods to improve this.

Causes of latency

The causes of latency fall into two categories. Derrien (2014) describes these as *algorithmic delay* and *computational delay*.

Algorithmic delay is intrinsic to the function of the algorithm. As an example, if an algorithm requires N samples within its analysis window in order to operate, the algorithmic delay will be at least N/F_S seconds (where F_S is the audio sampling frequency). Some algorithms use this window for comparison against the live signal stream (such as autocorrelation-based algorithms). This introduces additional algorithmic delay of $1/F_{MIN}$ seconds (where F_{MIN} is the lowest frequency the device is expected to be able to detect).

Computational delay is the amount of processing time the algorithm uses to process all the samples in the analysis window. The most obvious method to reduce this is to

use a more powerful processor. In some cases, there may also be ways to optimise the algorithm itself to reduce this delay. Such modifications could include restructuring code to optimise calculation stages, and reducing the number of computationally expensive multiplication/division operations required.

Musicians can detect latencies from between 20 ms and 30 ms (Pardue, Nian, Harte, & McPherson, 2012), and therefore, the latency of this device must fall below this threshold.

Algorithms and categorisation
Algorithms can be broadly categorised by the domain in which they operate: time domain, frequency domain, or hybrid methods utilising data from both domains.

Frequency domain algorithms
Frequency domain algorithms exploit the principle that periodic signals will produce spectral peaks at F_0 and its harmonics. Frequency magnitude data can be used directly, or the separation between peaks can be measured, since harmonics occur at integer multiples of F_0 (Upadhya, 2012). There are various well-documented frequency domain algorithms, such as Cepstrum (CEP) and Harmonic Product Spectrum (HPS) (Gerhard, 2003; Middleton, 2003). Since the accuracy of these algorithms largely depends on the precision of the data provided to them, the frequency domain transformation process must generate precise data without introducing excessive latency.

The transformation is achieved using Discrete Fourier Transform (DFT). A Fast Fourier Transform (FFT) is a more computationally efficient method of performing DFTs (thus reducing computational delay), but has the disadvantage of creating linear bin separations. Therefore, to achieve adequate resolution at lower frequencies, the narrow bin bandwidth would create an excessive number of bins in the higher frequencies, owing to the logarithmic nature of pitch perception, which consequently becomes computationally inefficient.

In addition, IRCAM (n.d.) recommend a window of time domain signal that is five times longer than the lowest detectable F_0 period to achieve accurate results from the frequency domain transformation. Since the human vocal pitch range extends down to F2 (Husband, 1999), which has a cycle period of 11.45 ms, the minimum window size that should be considered is 57.25 ms. This would introduce excessive algorithmic delay.

For these reasons, frequency domain algorithms (including hybrid algorithms) were deemed unsuitable for this application as it would be impossible to achieve the target latency, regardless of platform processing power.

Time domain algorithms
These algorithms fundamentally measure intervals between recurring markers within the waveform or detect repeating waveform patterns. The simplest and most computationally light algorithms are peak rate (PR) and zero crossing rate (ZCR). An ideal signal has just two zero crossing points per cycle separating very distinct maximum and minimum points. The period between these key points determines the frequency of the signal. However, vocal waveforms are usually much more complex

owing to other spectral content (as illustrated in Figure 1), making it difficult to distinguish the key points of the fundamental frequency component.

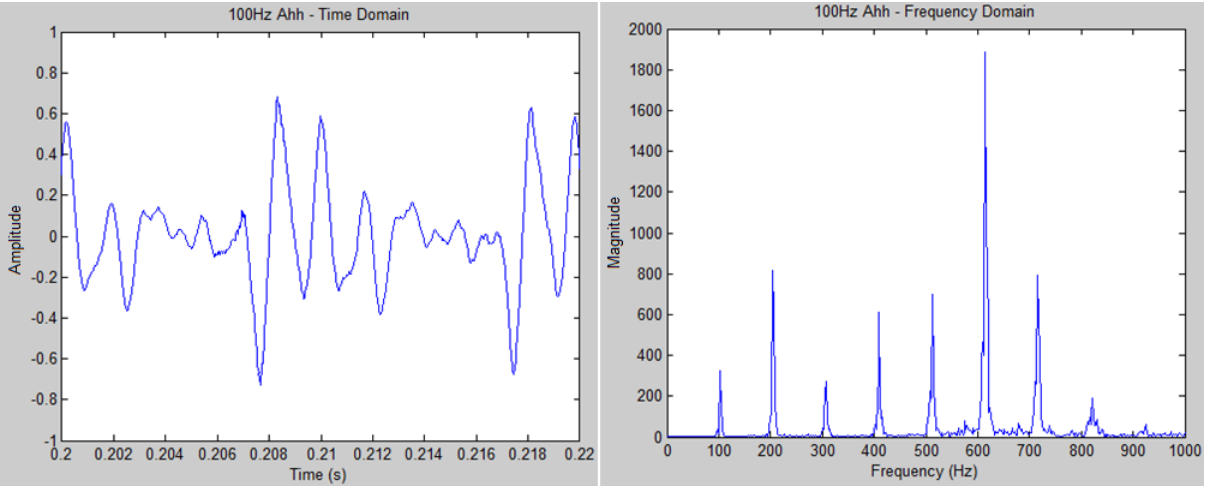

Figure 1: Vocalised 'Ahh' spectral content

Hysteresis could be employed to require the signal to swing a predefined threshold away from the pitch marker before monitoring for the next, but this would still fail with strong harmonics such as those in the example above. Therefore, PR and ZCR are unsuitable for this purpose. However, there is another family of algorithms operating within the time domain that are more capable of tolerating rich harmonic content. These algorithms are based around autocorrelation.

Autocorrelation is the process of taking a window of signal (referred to as the 'specimen' in this paper) and comparing the correlation with the original signal at various time offsets (hence the 'auto' prefix.) The similarity produces a correlation score that will peak when the lowest frequency component begins a new cycle (and thus all harmonics also realign) (Gerhard, 2003).

Autocorrelation (AUTOC) is defined by the equation shown in Figure 2, where *m* is the offset position from which to perform autocorrelation and *N* is the specimen size in samples.

$$R(m) = \frac{1}{N} \sum_{n=0}^{N-1-m} x(n) \cdot x(n+m)$$
Figure 2: AUTOC equation

In practice, the equation simply accumulates the result of multiplying each sample in the specimen with the corresponding sample in the original signal at the given offset position. Better alignment of the specimen and the signal at the offset position causes more samples to be multiplied with samples of the same polarity, thus producing a higher correlation score. An example of this is provided by Figure 3, where the blue line denotes the time domain signal stored in the buffer and the red line denotes the excerpt taken as the specimen.

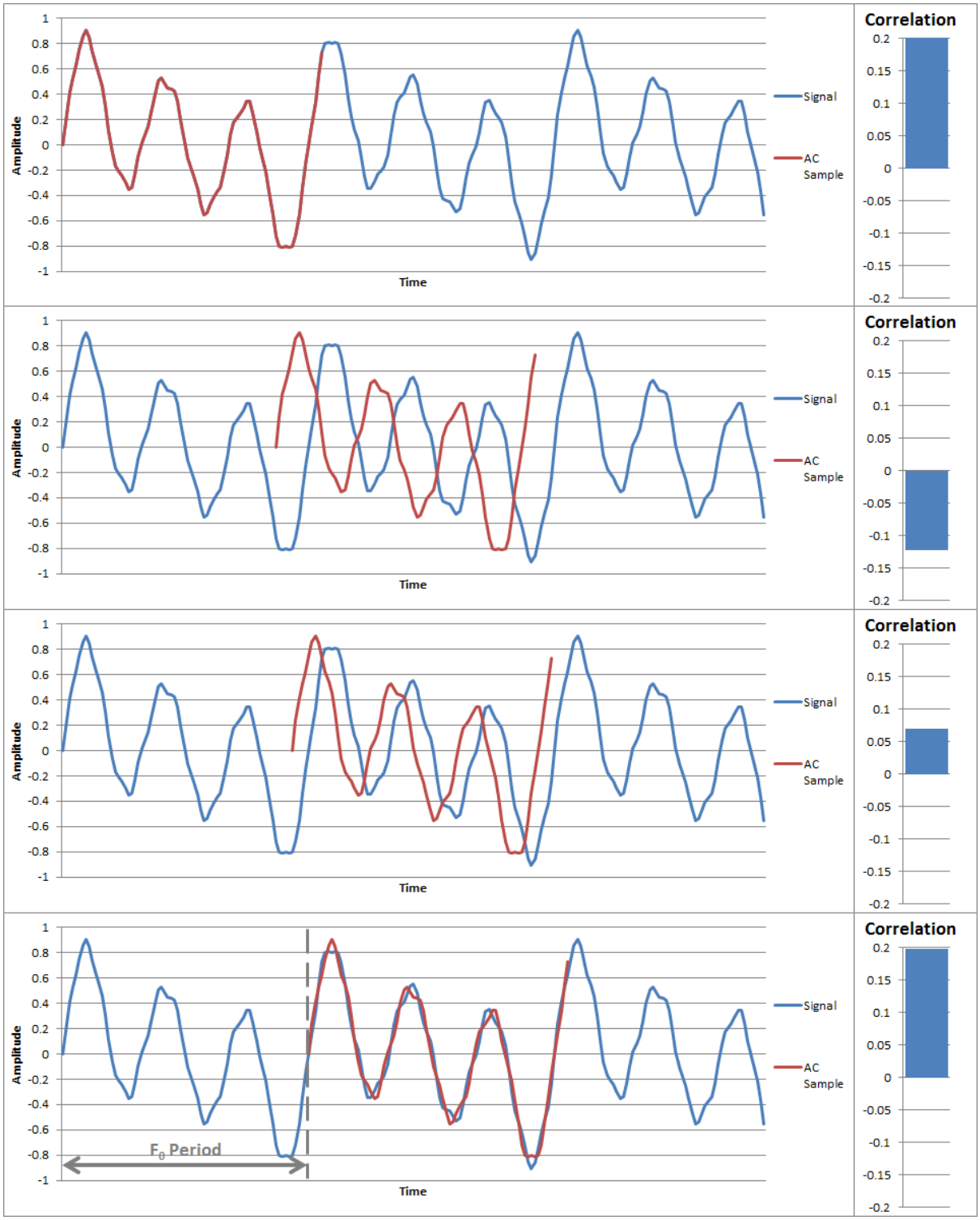

Figure 3: AUTOC demonstration

AUTOC provides the foundation for all other autocorrelation-based algorithms.

The modified autocorrelation function (MACF) mimics AUTOC but employs centre-clipping. Licklider and Pollack (1948) first used centre-clipping as a spectral flattening technique to reduce the effect of noise and formants in speech, effectively zeroing all samples within set boundaries and offsetting the remaining samples by the boundary value (demonstrated in Figure 4). Any sample that has crossed zero to

the opposite polarity as a result of noise will be prevented from creating a negative effect on the overall correlation score. Simply put, it ensures that the remaining samples either side of zero were intended to be of that polarity owing to their original distance from zero. The boundaries are a set percentage of the peak signal amplitude, varying from 30% (Upadhya, 2012) to 80% (Dubnowski, Schafer, & Rabiner, 1976) in different studies.

The infinite-peak-clipping autocorrelation function (IACF) follows the centre-clipping technique of MACF, but samples outside of the centre-clipping boundaries are thrown to the amplitude extremities, as shown by Figure 4. The samples essentially become tri-state.

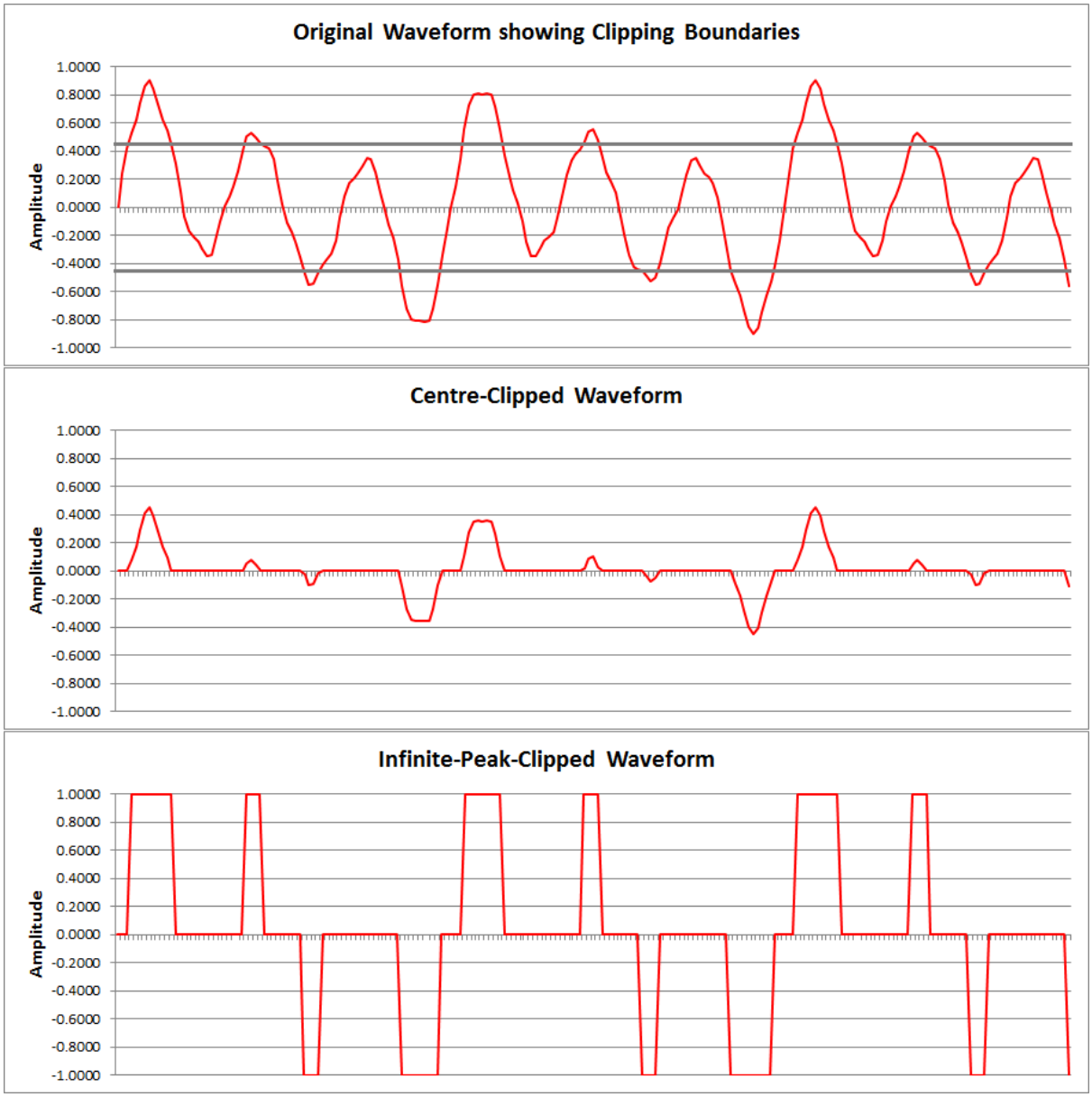

Figure 4: Centre-clipping and infinite-peak-clipping

The average magnitude difference function (AMDF) closely resembles AUTOC but uses the magnitude difference between samples by finding the absolute value of

their difference using the equation in Figure 5. Therefore, a lower score is considered a better correlation match with AMDF.

$$R = \frac{1}{N}\sum_{n=0}^{N-1}|x(n) - y(n)|$$

Figure 5: AMDF equation

This is intended to reduce computational delay over AUTOC since subtraction is computationally less expensive than multiplication.

Implementation

The four autocorrelation-based algorithms selected in the previous section were implemented in the C++ programming language on a Windows PC. Various parameters define how these algorithms operate, and these were made configurable via command-line arguments.

Frame size: This defines length of the frame containing the audio specimen expressed relative to the minimum detectable wavelength (since it would be feasible to assume at least one full wavelength would be necessary for an accurate match). The relationship between the frame size and the overall buffer length is demonstrated by Figure 6. A larger frame provides more data for correlation, which could theoretically improve accuracy at the expense of further computational delay.

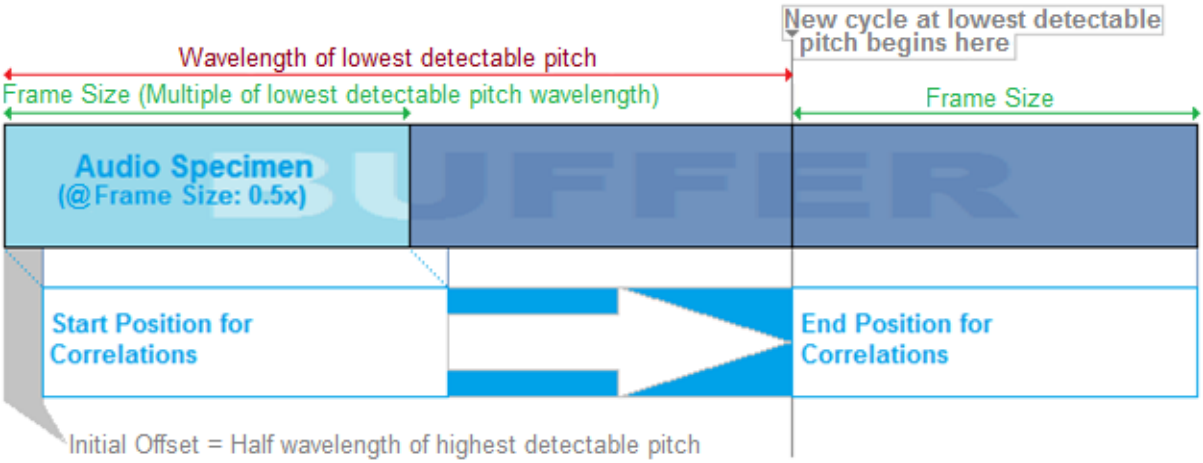

Figure 6: Buffer length to frame size and detectable pitch relationship

Sampling frequency (F_s): As the pitch of a signal increases, each cycle contains fewer samples. This imposes a limit at which semitonal differences are no longer distinguishable. Increasing the sampling frequency improves time resolution and consequently raises this upper limit. However, this also requires more buffer memory and more processing time to analyse and, therefore, lower sampling frequencies are more favourable in terms of computational latency.

A 15625 Hz sampling frequency is required in order to maintain semitonal distinction up to note A5. However, the sampling frequency can be lowered if high notes are not going to be utilised in order to improve latency. Therefore, sampling frequencies down to 8000 Hz were configurable. Finally, 48 kHz was included as an option in

order to assess the potential benefits that high sampling rates might offer if a sufficiently powerful platform was used.

Required accuracy: Expressed as a percentage of the best possible correlation score, this provided a threshold that a correlation score had to meet in order to be deemed a credible detection and trigger an output. The best possible score is the correlation score at zero lag, where the specimen is completely aligned with the original signal. For most algorithms, this produces the highest possible correlation score. For a difference function, however, this produces a score of zero and the worst plausible score occurs when the waveform is completely inverted. Therefore, in this case the percentage defined the point within the range of worst possible score to zero. This threshold not only controlled the required confidence in a result before allowing output, but was also designed as a voiced/unvoiced (V/UV) filter. Unvoiced sounds in speech generally have a very short duration and do not exhibit discernible recurring waveform patterns, consequently producing poor correlation scores that can be rejected by the required accuracy mechanism.

Centre-clipping boundary (for MACF and IACF): This was expressed as a percentage of the lower of the absolute positive and negative waveform peaks following the methodology of Dubnowski et al. (1976).

The following parameters were fixed in software:

Gate threshold: This was fixed at approximately 16% of the maximum possible amplitude swing. The algorithm would only run on the buffered audio if it breached this threshold. This was to prevent background noise from being analysed for pitch. Additionally, gate closure was also used to trigger MIDI *note off* messages.

Clip threshold: This was also fixed in software at approximately 98%. It had no influence on algorithm behaviour. It was simply implemented to inform the user of potential clipping by displaying a warning.

Accuracy improvements
Initial trials revealed a higher than anticipated number of incorrect detections occurring, regardless of which algorithm and parameter combination was specified. The causes of these incorrect detections were assessed and corrective features were implemented to handle these occurrences.

Low amplitude signals bordering the gate threshold could cause a gate chatter effect, resulting in many MIDI *note on* and *note off* messages being sent rapidly in succession. An iteration counter was implemented to rectify this, requiring a set number of concurrent audio specimens to fail to exceed the gate threshold before sending the *note off* message. The limit was set to 50 in code.

Similarly, if the vocalised tone bordered the boundary of two notes, minor pitch fluctuations could cause the output to alternate rapidly between the notes. This was more apparent with higher notes where a cycle would consist of fewer samples. Higher sampling frequencies may improve accuracy by increasing time resolution, but would also introduce more computational delay. Furthermore, this would still not resolve the jitter caused if the vocalist sang flat and bordered two notes. Therefore,

consistency-checking counters were implemented to assess note stability. Two parameters specified these periods. The first defined the number of consecutive, consistent results required before allowing a *note on* message to be sent. The second parameter specified the number of consistent results required to permit a note change when a note was already playing. These values were set at 5 and 20 respectively. The first value was intentionally set lower to reduce note onset latency.

Since the human voice is not a perfect tone generator (i.e., cycles of a vocalised tone will not perfectly match each other), it is possible that subsequent repetitions of the fundamental cycle correlate better with the specimen than the first occurrence. A simplified version of this problem is shown in Figure 7. The subsequent occurrence of F_0 achieves the greatest correlation score and is therefore used to estimate the vocalised pitch rather than the first occurrence. Consequently, the algorithm outputs the note an octave lower than expected.

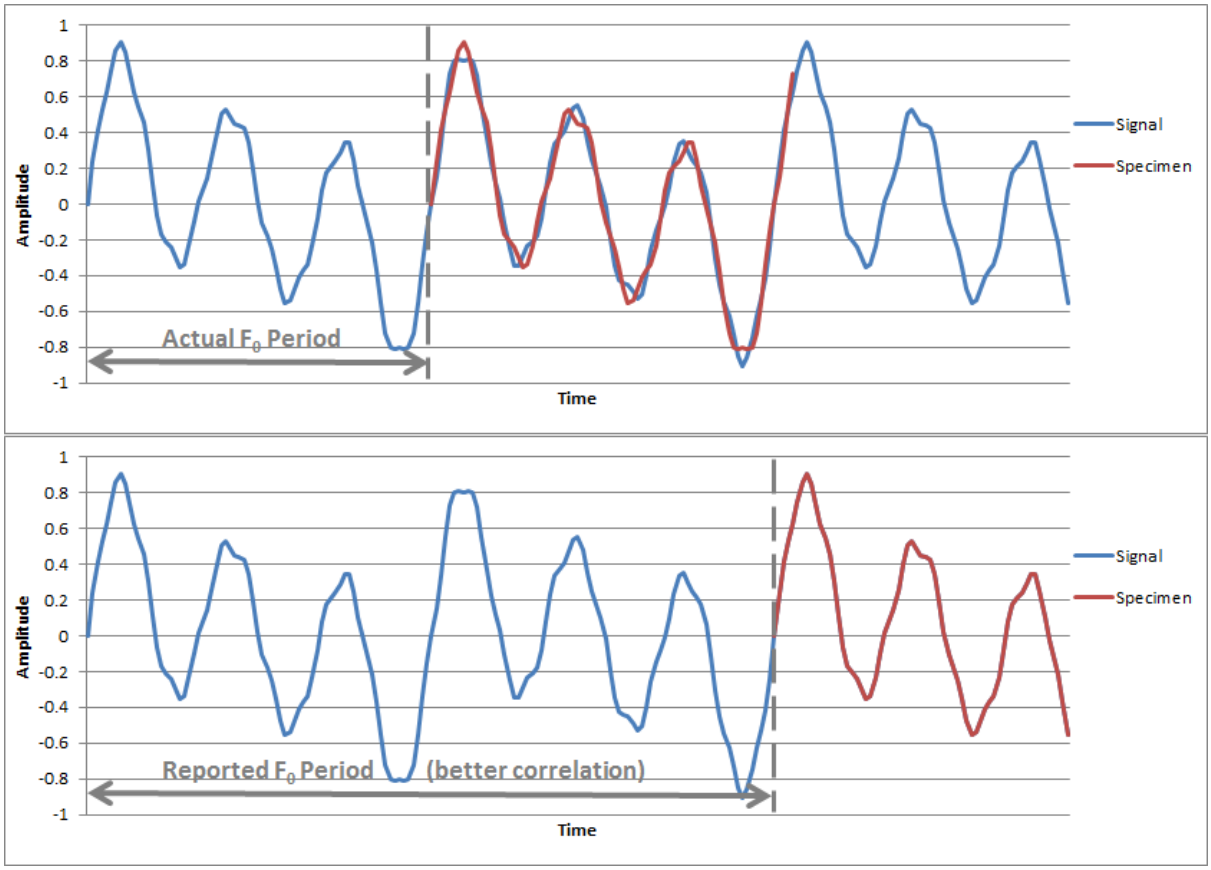

Figure 7: Octave error

An improvised 'Peak Target' feature was implemented that defined the amount of correlation improvement required for subsequent correlation peaks to override the current best score. This was a configurable parameter in order to determine which value reduced octave error most effectively through testing. The optimal value would improve accuracy by ensuring that, even if a subsequent repetition of the fundamental cycle achieved a better correlation score than the first occurrence, it would still be rejected, as it would not reach the target required. However, if set too high, the correlation score at the true fundamental would not be sufficient to override the scores achieved at earlier lag periods where harmonic frequencies began to

realign. This would produce the opposite problem, in which the reported pitch was higher than expected.

Speed optimisations
Various optimisations were implemented to reduce computational delay. Since the intended platform was an embedded system, it was particularly important that the software ran as efficiently as possible to minimise the hardware performance requirements necessary to run the software adequately, thus reducing hardware cost.

The gate trigger and MIDI note velocity value should use a root-mean-squared (RMS) measurement across the buffer. However, the software alternatively found the peak sample amplitude within the buffer, since this is far less computationally expensive.

Eight-bit samples were used, since higher bit-depths would increase computational latency, particularly on an embedded system that may have a narrow data bus. The use of integers to represent samples avoided floating-point mathematics and reduced buffer memory requirements by 75%.

The averaging operation of each correlation function was omitted, avoiding an unnecessary division operation. Since the number of samples processed in each correlation analysis remained consistent, the scores were already directly comparable.

An array was prepopulated as an optimised lookup table for mapping sample offsets to MIDI note numbers using the formula in Figure 8 (where *x* is the MIDI note number and *n* is the offset.)

$$x = 12 \log_2 \left(\frac{F_s}{440n} \right) + 69$$

Figure 8: Offset to note number equation

Logic operations were used in algorithms where possible to reduce the number of mathematical operations. MACF latency was reduced by approximately 20% by zero-checking samples after centre-clipping. IACF latency was reduced by 68% as all sample correlations could be handled by logic, since the signal essentially becomes tri-state.

Test methodology
Testing was performed using an Intel Celeron 1.86 GHz laptop running Windows XP 32-bit. In order to provide consistent test conditions, a series of audio clips were used in testing rather than live signals. This also meant the process was completely automated, which enabled thorough testing and the compilation of an extensive results set.

Test subjects
One hundred and sixteen samples of single vocalised notes were collated and edited to remove any leading silence to provide accurate latency measurements. Each

sample was analysed for pitch using various correlation and FFT analyses in the *Audacity* audio editing software. Slight vibrato in the sample set was permissible, provided that it did not vary by a semitone or more.

Although the software was originally intended to analyse hummed melodies, it was hoped that the correct parameter set would allow the algorithms to tolerate various transients. Therefore, the sample set also included non-lexical vocables such as 'Ooh', 'Ahh', 'Laa', 'Ooo', 'Mee', 'Haa', 'Eee' and 'Moo'. The samples also varied between male and female, ranged in pitch from F2 to A5, and ranged in vocal timbre from very smooth to harsh and croaky.

Algorithm parameters

Four stages of testing were conducted, with each stage aiming to seek out the optimal parameters based of the results of the previous stage. A total of 636 tests were conducted, covering various combinations of the parameter values in Table 1.

Parameter	Values
Algorithm	*All*
Frame Size	0.5x, 0.6x, 0.7x, 0.8x, 1.0x, 1.5x, 2.0x, 2.5x
Sampling Frequency	8000 Hz, 12500 Hz, 15625 Hz, 48000 Hz
Centre-Clipping	10%, 20%, 30%, 35%, 40%, 70%
Required Accuracy	60%, 80%, 85%, 90%, 95%
Peak Target	0.0%, 4.0%, 5.0%, 5.5%, 6.0%, 7.0%, 7.5%, 8.0%, 9.0%

Table 1: Algorithm Parameters

A range of frame sizes were tested from 0.5x to 2.5x. Particular attention was given to the lower values, since this would reduce both algorithmic and computational delay.

A sampling frequency of 15.625 kHz was trialled as this was the lowest frequency at which semitonal differences are still distinguishable up to A5. Frequencies of 12.5 kHz and 8 kHz were trialled, although these limited the maximum detectable note to F5 and C5 respectively. A frequency of 48 kHz was also trialled to investigate the benefits that greater sampling rates may provide, given a suitably capable platform.

The first stage of testing revealed that high centre-clipping boundaries did not perform well, and therefore subsequent stages focused on the lower end of the range. Similarly, early tests revealed that lower 'Required Accuracy' and 'Peak Target' parameters did not perform well, and so investigation then focused on higher values.

Automated test execution

A script was written to generate all combinations of given parameters for each stage of testing and write them to a comma-separated-values (CSV) file. A separate application then read the file and ran the pitch detection software for each test with the correct parameters by using the relevant command-line arguments.

Software modification
The pitch detection software had to be modified to read samples from audio files rather than from the system audio buffer. A high-precision timer was reset when the first test subject was loaded. Each time the software requested new samples, the timer was queried to determine where in the audio file to extract samples from, mimicking a live, real-time signal. The samples were copied into the buffer while also compensating for sampling frequency differences. Once the end of the audio file was reached, the next test subject was loaded.

Each time a new test subject was loaded, the details were logged to a results file to record which test subject was in use when MIDI events occurred. As such, MIDI note functions were also modified to write their output to the results file with a timestamp (relative to the start of each audio clip) for each event.

Results analysis
A script was written to analyse the data recorded by pulling the raw data from the results file of each test, analysing the MIDI data, and populating a spread sheet with various useful statistics such as (but not limited to):

- percentage of times the first note played was the correct note;
- percentage of note-on time spent playing the correct note;
- average latency between sample playback and first note on, and the correct note;
- average number of note changes during sample playback (fluctuations).

Findings
The IACF algorithm triumphed in terms of both accuracy and latency, as shown by Figure 9. In this graph, each data point represents a complete test with measurements averaged across all test subjects. Each test was based on a different combination of parameters.

The graph is plotted as latency against accuracy, since these are the two key factors determining overall performance. Latency was based on the average delay between the start of each audio sample and the output of the correct note. Accuracy was represented by the average percentage of time spent outputting the correct note for each test subject. It did not include the silence before the first note result was acquired, since this is accounted for by the latency axis. Furthermore, this measure also inherently considers note fluctuations, semitonal error and octave error, since any time spent outputting the wrong note would consequently reduce the accuracy percentage.

The latency improvement witnessed using IACF was expected, since it is the only algorithm to use purely logic operations over mathematics (aside from the accumulation of sample correlations).
With the primary goal of the study being to achieve minimal latency while achieving satisfactory accuracy, it is clear from Figure 9 that assessment of parameter influence had to be focused around IACF.

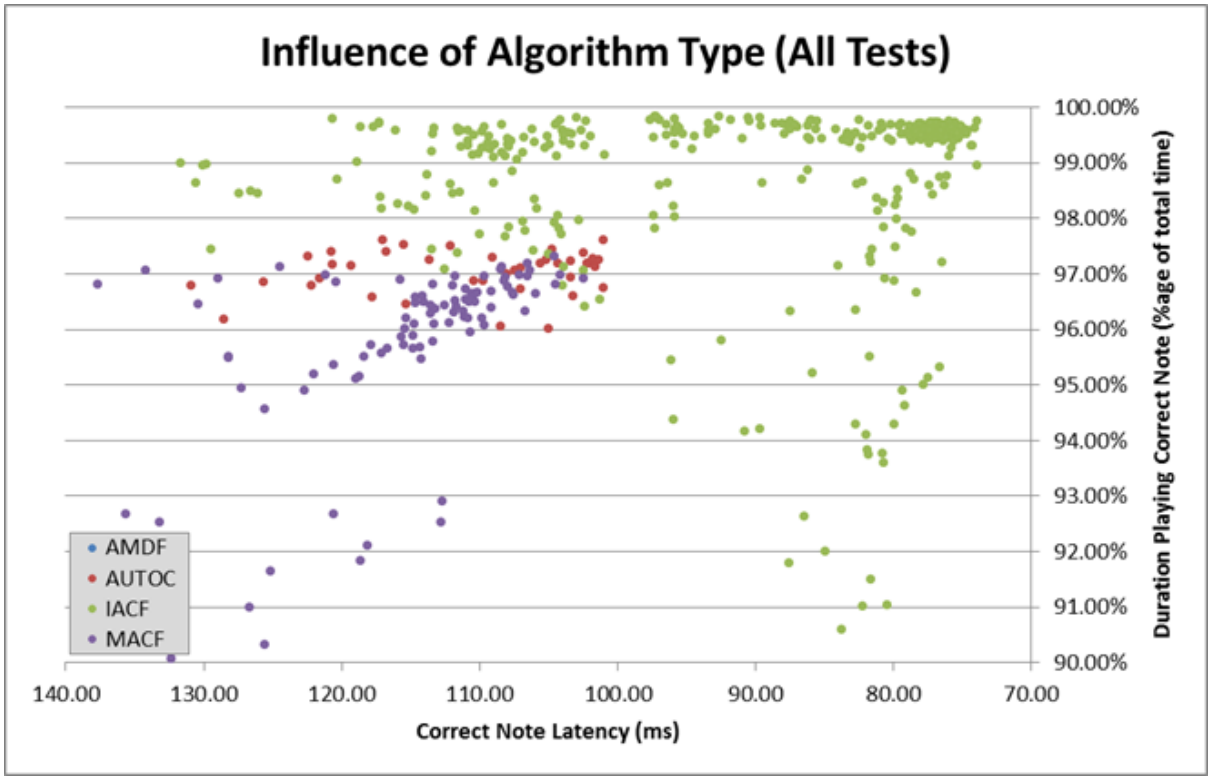

Figure 9: Algorithm performance

The only discernible trend in IACF accuracy influenced by peak target is that values of 5.0% or below performed more poorly than those above, as shown in Figure 10. No obvious trend forms towards any particular value in the range of 5.5% to 9.0%. Again, each data point in Figure 10 represents one complete test.

Equally, the only discernible accuracy trend for the centre-clipping parameter is that boundaries set above 40% cannot achieve accuracy ≥99%, as shown by Figure 11. However, there is a clear trend indicating that 85% required accuracy performed better and more consistently than any other value.

Platforms with greater processing capability could facilitate greater sampling frequencies and larger buffers. The potential benefit that this may yield was investigated. The results revealed that larger frame sizes could increase achievable accuracy (+0.09% between 0.5x and 2.5x) but would dramatically increase latency (+28.8%). This comparison was performed between the two tests indicated by the blue data points in Figure 12.

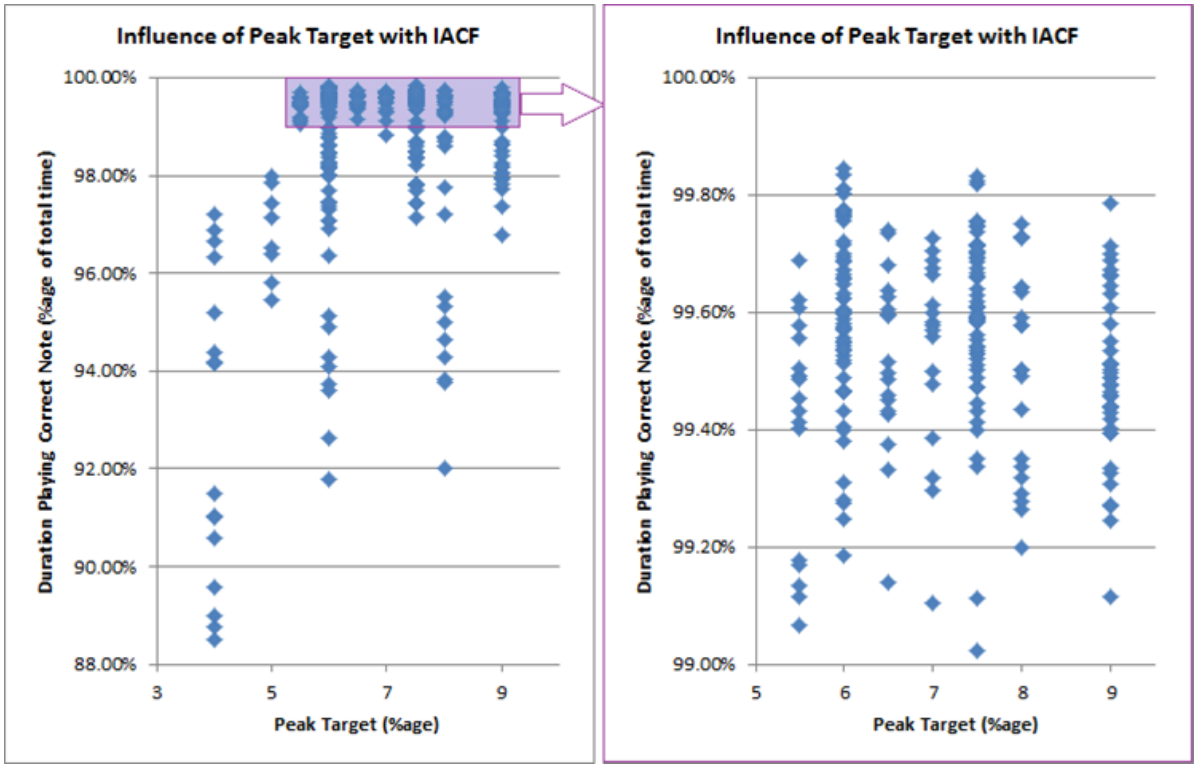

Figure 10: Peak target

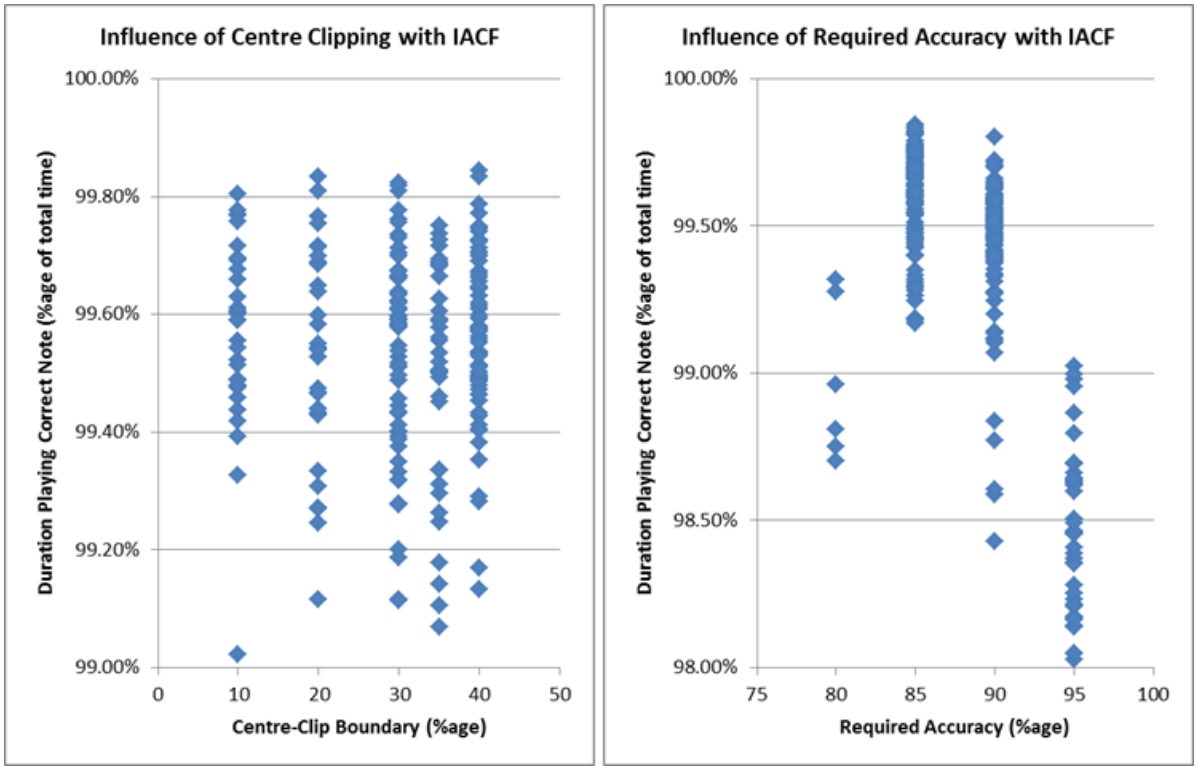

Figure 11: Centre-clipping boundary and required accuracy parameter

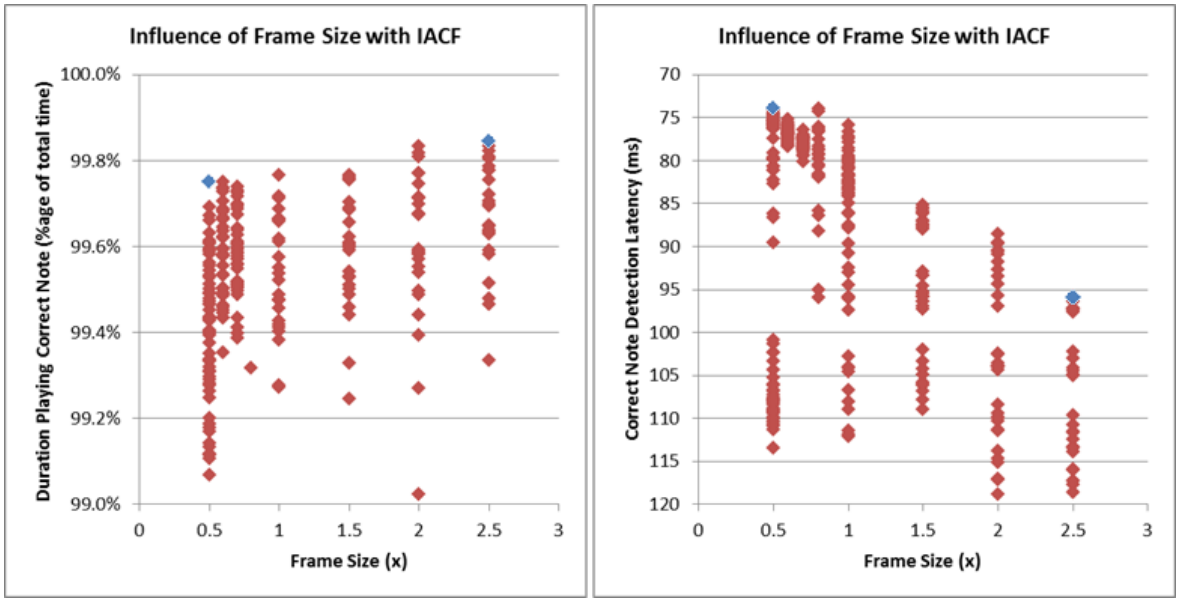

Figure 12: Frame size

Additionally, a 48 kHz sampling rate can offer a 0.51% increase in accuracy with just a 0.04% increase in latency on the test platform, as shown in Figure 13. However, it should be considered that the processing capabilities of an embedded system are likely to be far more limited than the test platform and, consequently, the total latency is likely to be greatly affected by the increased sampling frequency.

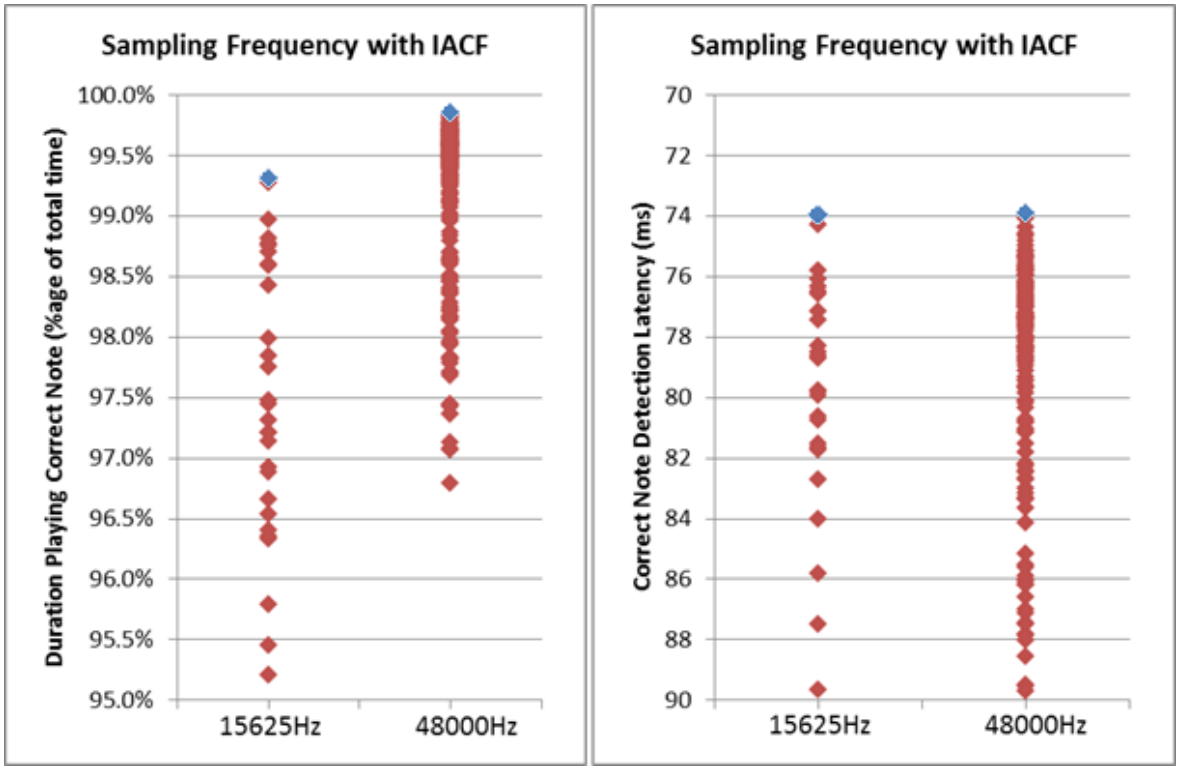

Figure 13: Sampling frequency

Analysis of individual test subjects across all tests conducted at the 15.625 kHz sampling frequency showed a steep decline in the correct note detection rate against pitch. This was expected owing to the logarithmic nature of pitch perception. As pitch

increases, fewer offsets are assigned to each note and therefore, any slight deviation would select the wrong note.

At 48 kHz, the vocal pitch limit of D6 (Husband, 1999) still has four offset positions assigned to it and algorithms could still theoretically distinguish semitonal differences up to C8. A 15.625 kHz sampling frequency limits the maximum detectable pitch to A5, and notes F#5 to A5 have just one offset position assigned to them at this rate. Consequently, using a 48 kHz sampling rate over 15.625 kHz delivers a dramatic improvement in the correct note detection rate at higher pitches.

In Figure 14, each data point represents one of the 116 individual test subjects, with its detection rate determined by the average percentage of time that algorithms output the correct note across all tests executed at that sampling frequency.

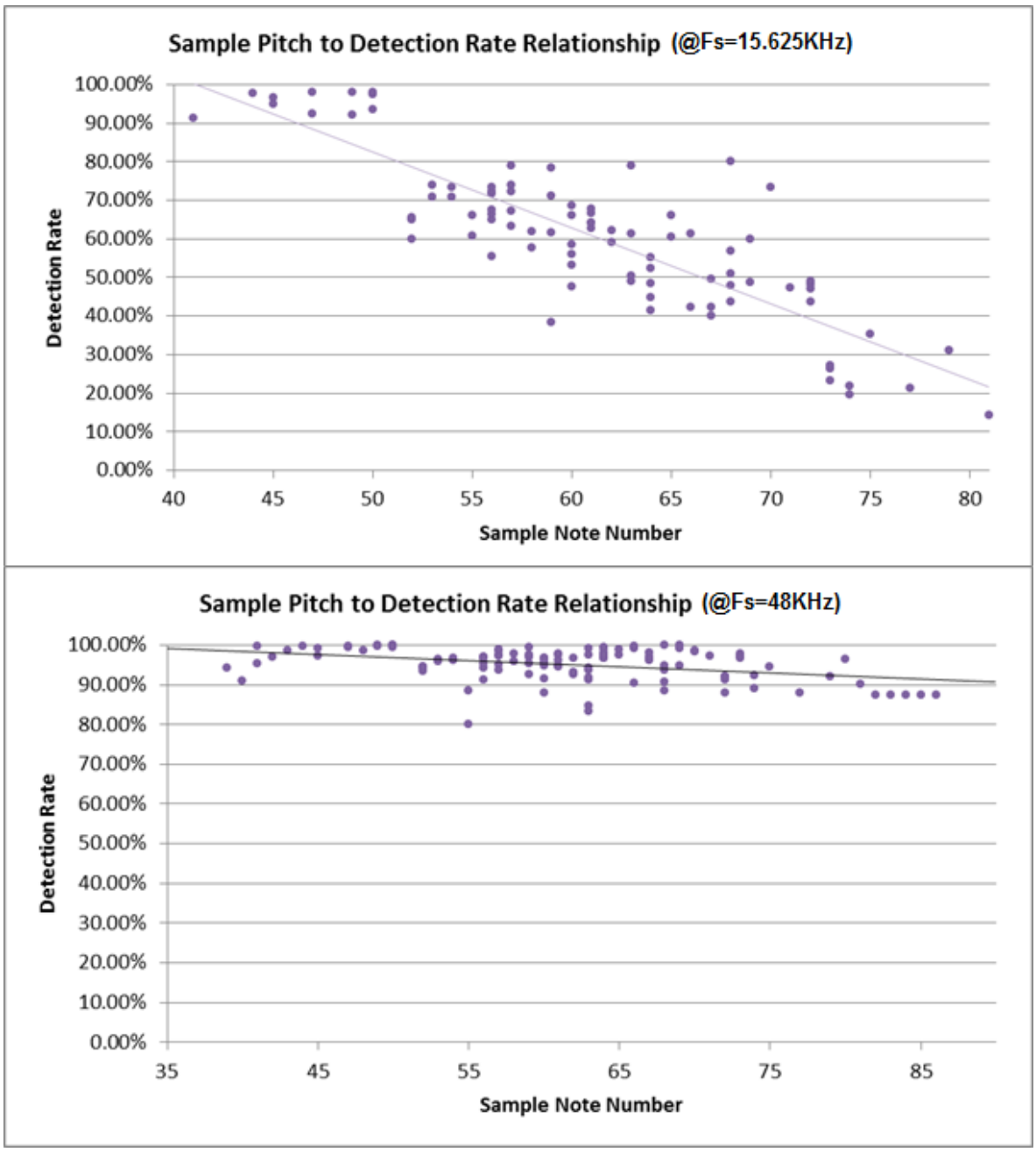

Figure 14: Pitch detection rate

A table of the ten best-performing parameter settings was generated based on the following criteria:

- correct note latency: <100ms
- instances of first note being the correct note: >98%
- sorted by duration with correct note.

The IACF algorithm was in use during every one of these tests because of its significant latency advantage while still achieving excellent result accuracy. All of these tests were at 48 kHz sampling frequency, as expected. Results from tests conducted at 48 kHz sampling frequency and those at lower frequencies emphasised the importance of sampling rate (and thus platform processing performance), particularly for the accurate detection of higher notes. Additionally, centre-clipping ranges from 10% to 40%, peak targets are within the range 5.5–9.0%, and required accuracy is always 85%, as previously suggested by Figures 10 and 11.

The superior accuracy provided by the IACF algorithm may be a direct consequence of its speed advantage. The more algorithm iterations that can be run within a given time period, the more confidence there can be in the result. In addition, the interval between start positions within the waveform for each execution of the algorithm would be reduced. For example, IACF using a 1.0x frame size at 48 kHz had an average computational delay of 3.6 ms (as measured by timers built in to the software). Since the buffer length is 25 ms at this frame size, there was 85.6% overlap between tests. The minimum latency achievable would be equal to the computational delay multiplied by the number of consecutive results required to pass consistency checks, plus algorithmic delay (buffer length). In this example, it is (3.6*5) + 25 = 43 ms, as explained by Figure 15.

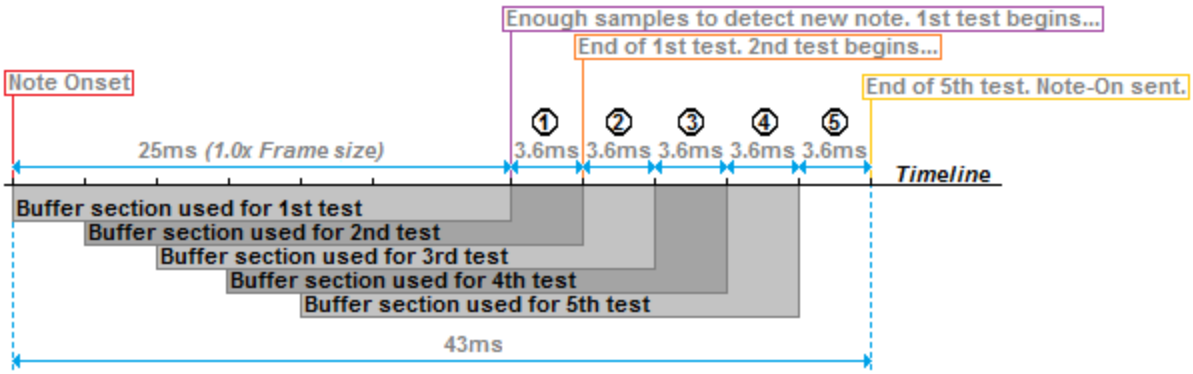

Figure 15: Consistency checking

Halving the frame size reduces the buffer length to 18.75 ms (as described by Figure 6) and the computational delay to 2.22 ms per iteration on the test platform (as measured by high-precision timers implemented in code). With five consistent results required to pass the initial note consistency-check, a total latency of 29.85 ms should be theoretically possible, but even the 43 ms latency conceived in Figure 15 could not be achieved.

Table 2: Best Performance Settings

Algorithm Settings							Instances of First Note Correct		Duration Playing Correct Note			
Algorithm	Frame Size	Sampling Frequency (Hz)	Centre-Clipping (%)	Required Accuracy (%)	Peak Target (%)	Correct Note Latency (ms)	Correct Only	Ignoring Octave Error	Correct Only	Ignoring Octave Error	Ignoring Semitone Error	Average Number of Note Changes
IACF	2.0	48k	20	85	6.0	92.65	98.28%	98.28%	99.83%	100.00%	99.83%	0.38
IACF	2.5	48k	30	85	7.5	97.17	98.28%	98.28%	99.82%	99.98%	99.82%	0.37
IACF	2.5	48k	40	85	9.0	95.82	98.28%	98.28%	99.79%	100.00%	99.79%	0.29
IACF	2.0	48k	10	85	6.0	96.95	99.14%	99.14%	99.77%	100.00%	99.77%	0.39
IACF	0.5	48k	40	85	7.5	73.92	98.28%	98.28%	99.75%	99.97%	99.76%	0.84
IACF	1.0	48k	20	85	7.5	79.44	98.28%	98.28%	99.72%	99.99%	99.73%	0.59
IACF	2.5	48k	30	85	9.0	96.44	99.14%	99.14%	99.70%	100.00%	99.70%	0.37
IACF	2.0	48k	10	85	7.5	95.67	98.28%	99.14%	99.68%	99.99%	99.68%	0.45
IACF	2.0	48k	30	85	9.0	89.55	99.14%	99.14%	99.67%	99.98%	99.69%	0.44
IACF	1.5	48k	30	85	9.0	85.61	98.28%	98.28%	99.61%	99.98%	99.61%	0.47

This was discovered to be due to note onset slew rather than failings of the algorithm itself. When a note is sung, it is rarely sung at the correct pitch immediately. Analysis of a variety of test subjects using the *Melodyne* pitch-correction software revealed that many had a pitch bend at the beginning of the note (as indicated by the red line in Figure 16). This bend typically spans one or two semitones, but occasionally more. Consequently, consistency checks would continually fail until the pitch stabilised on a single note.

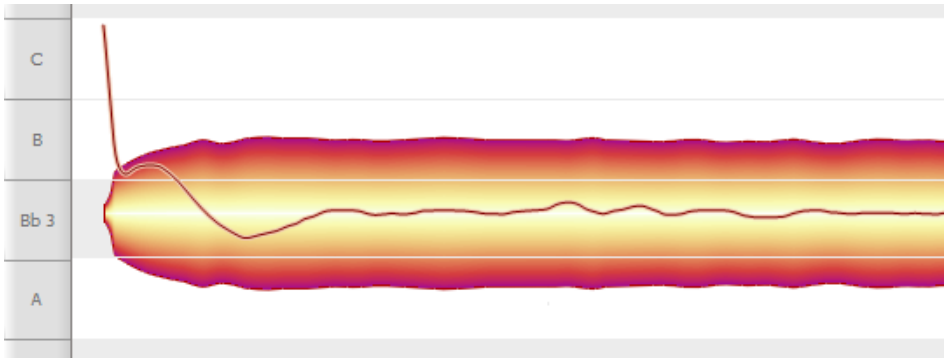

Figure 16: Note onset slew

Conclusions

Several conclusions can be drawn from the findings of the testing process regarding the implementation of autocorrelation-based algorithms for low latency pitch detection in live signal applications.

The most significant conclusion is that IACF outperforms AUTOC, AMDF and MACF in terms of both accuracy and latency. AMDF performed the worst, with no parameter combination able to achieve >90% accuracy with <140 ms latency with this algorithm. Another key conclusion is that oversampling considerably improves accuracy. Despite the algorithm being theoretically able to detect semitonal differences up to note A5 when using a 15.625 kHz sampling rate, tripling the sampling frequency to 48 kHz substantially improved detection accuracy for higher notes.

When observing the effects of specific algorithm parameters, it was apparent that the 'Peak Target' feature reduced octave error significantly, although some instances still occurred, as evident in the accuracy value differences between 'correct note only' and 'ignoring octave error' in Table 2. Ideal values for peak target were found between 5.5% and 9.0%. It was also concluded that 'Required Accuracy' functions optimally at 85%. Higher values begin to reject correct detections and lower values begin to accept incorrect detections. The ideal centre-clipping boundary value was less conclusive, with values in the relatively broad range of 10% to 40% usually producing the most accurate results. As anticipated, larger frame sizes tend to yield more accurate results, but also increase latency. Interestingly, however, even 0.5x frame size can achieve satisfactory performance given the correct parameter combination, as proven by the fifth best performing of 636 tests in Table 2.

Although it was deemed theoretically possible to achieve an overall latency of less than 30 ms, investigation concluded that note onset slew in vocalised tones was the

cause of extended detection delay owing to pitch instability, and thus the reason that the target latency was not achieved.

Further work

Since initial note error or extended latency is usually due to note slew rather than erroneous detection, device behaviour could change depending on the application.

Live mode: The device requires a lower number of consistency-checking iterations to provide minimal onset latency. As the note slews, the device sends MIDI pitch bend messages to follow the input signal (with interpolation). This would only function well for sustained patches without strong transients. Otherwise, it would be easily noticeable that the transient was played at the wrong pitch. Validity checking would also need to be implemented to ensure that subsequent results are not implying slews greater than a few semitones.

Composition mode: A message buffer is used to ensure that all MIDI data is delayed by the same amount (which must be the longest latency likely to be encountered or greater). The output can then simply be realigned in the receiving DAW.

References

Derrien, O. (2014). A very low latency pitch tracker for audio to MIDI conversion. In *17th International conference on Digital Audio Effects, Erlangen, Germany*. Retrieved from http://www.dafx14.fau.de/papers/dafx14_olivier_derrien_a_very_low_latency_pitch_.pdf

Dubnowski, J. J., Schafer, R. W., & Rabiner, L. (1976). Real-time digital hardware pitch detector. *IEEE Transactions on Acoustics, Speech, and Signal Processing, 24*(1), 2–8. http://dx.doi.org/10.1109/TASSP.1976.1162765

Gerhard, D. (2003). *Pitch Extraction and Fundamental Frequency: History and Current Techniques*. Technical Report TR-CS 2003-06, University of Regina, Canada. Retrieved from: http://www.cs.uregina.ca/Research/Techreports/2003-06.pdf

Husband, G. (1999). *What's in your Music?* Retrieved from http://www.tnt-audio.com/topics/frequency_e.html

IRCAM. (n.d.). *FFT Parameters - Window Size*. Retrieved from http://support.ircam.fr/docs/AudioSculpt/3.0/co/Window%20Size.html

Licklider, J. C. R., & Pollack, I. (1948). Effects of Differentiation, Integration, and Infinite Peak Clipping upon the Intelligibility of Speech. *Journal of the Acoustical Society of America, 20,* 42–51. http://dx.doi.org/10.1121/1.1906346

Middleton, G. (2003). *Pitch Detection Algorithms*. Retrieved from: http://cnx.org/contents/8b900091-908f-42ad-b93d-806415434b46/Pitch-Detection-Algorithms

Pardue, L., Nian, D., Harte, C., & McPherson, A. (2014). Low-Latency Audio Pitch Tracking: A Multi-Modal Sensor-Assisted Approach. In *Proceedings of the International Conference on New Interfaces for Musical Expression* (pp. 54–59).

Upadhya, S. S. (2012). Pitch detection in time and frequency domain. In *2012 International Conference on Communication, Information & Computing Technology* (pp. 1–5).

http://dx.doi.org/10.5920/fields.2016.2125