2008-06-01

A Real-time Framework for Video Time and Pitch Scale Modification

Ivan Damnjanovic
Queen Mary University London

Dan Barry
Technological University Dublin, dan.barry@tudublin.ie

David Dorran
Technological University Dublin

See next page for additional authors

Recommended Citation

Damnjanovic, I. et al. (2008) A Real-Time Framework for Video Time and Pitch Scale Modification. Proc. of the *11th International. Conference on Digital Audio Effects (DAFx-08)*, Espoo, Finland, September 1-4, 2008.

Funder: European Community

A Real-time Framework for Video Time and Pitch Scale Modification

Ivan Damnjanovic, Dan Barry, David Dorran, and Joshua D. Reiss

***Abstract*- A framework is presented which addresses the issues related to the real-time implementation of synchronised video and audio time-scale and pitch-scale modification algorithms. It allows for seamless real-time transition between continually varying, independent time-scale and pitch-scale parameters arising as a result of manual or automatic intervention. We illuminate the problems which arise in a real-time context as well as provide novel solutions to prevent artefacts, minimise latency, and improve synchronisation. The time and pitch scaling approach is based on a modified phase vocoder with optional phase locking and an integrated transient detector which enables high quality transient preservation in real-time. A novel method for audio/visual synchronisation was implemented in order to ensure no perceptible latency between audio and video while real-time time scaling and pitch shifting is applied. Evaluation results are reported which demonstrate both high audio quality and minimal synchronisation error.**

***Index Terms*—Time scale modification, Audio/visual synchronisation, adaptive video refresh rate**

I. Introduction

Synchronised audio and video time stretching is often used in video editing and production whenever video content needs to be sped up or slowed down either as a creative effect or to fit certain time slots within a programme schedule, as is the case in television advertisements.

Time-scale modification (TSM) is typically used to change the tempo of musical content or the playback rate of speech without affecting pitch content. Conversely, pitch-scale modification (PSM) algorithms enable pitch shifting without affecting the playback rate of the audio content. A significant amount of research has been dedicated to both TSM and PSM yielding a variety of time and frequency domain algorithms. Despite this abundance of literature and readily available commercial applications, there is still a lack of information, understanding and consideration for real-time implementations of TSM and PSM algorithms. Here we illuminate some of the problems which arise in a real-time context as well as provide novel solutions to these issues. A real-time software based framework is presented, which allows time stretching of audio content within digital video streams whilst maintaining synchronisation with the video content. Time-scale changes can be made in real-time with almost unperceivable latency and no transitional artefacts. In addition, the approach also supports real-time pitch shifting of the audio content independent of time-scale changes. The approach is based on a modified phase vocoder with optional phase locking and an integrated transient detector which enables high quality transient preservation in real-time.

Within this article, emphasis is given to audio/visual synchronisation issues which arise in such a framework. Despite the growth in algorithms for independent audio time or pitch modification, there are relatively few applications which address combined time stretching of video and audio. In [1], a method for adjusting video playback rate to compensate for network delay is presented. Similarly, [2] presents an adaptive method for video playback, intended to address issues concerning packet loss and random delays in streaming applications. Their method uses audio time scaling when the streamed video playback speed is modified, as suggested for packet loss in voice communication [3].

Synchronised audio and video time scaling is typically used in video editing and production whenever video content needs to be sped up or slowed down either as a creative effect or to fit certain time slots within a programme schedule. For example, TSM can be used to alter the duration of an advertisement whilst preserving the pitch and timbre of speech and other audio content. Experiments have shown that increasing the information rate in commercials is more engaging and more favourable to viewers. In [4], it was suggested that an increase in the rate of information of up to 130 percent of the typical speech rate can significantly increase the impact of advertisements.

The driving force for the work presented here on real-time synchronised audio/video time-stretching comes from user requirements and user feedback in music education research [5, 6], which indicated that time-scaled video would be desirable in applications related to aural learning, music transcription and musical technique analysis. The effects of audio/video time-compression and expansion on the learning

Manuscript received June 11, 2009. This work was supported in part by the European Community under the Information Society Technologies (IST) programme of the 6th FP for RTD - project EASAIER contract IST-033902.

I. Damnjanovic is with Queen Mary U. of London, London, E14NS, UK (telephone: +44-2078827880, e-mail: ivan.damnjanovic@elec.qmul.ac.uk).

Dan Barry. Author is with the Audio Research Group in the Dublin Institute of Technology, Kevin Street, Dublin 8, Ireland (telephone: +353 1 4022862, e-mail: dan.barry@dit.ie).

David Dorran. is with the Audio Research Group in the Dublin Institute of Technology, Kevin Street, Dublin 8, Ireland (telephone: +353 1 4024873, e-mail: david.dorran@dit.ie).

Josh Reiss is with Queen Mary University of London, London, E14NS, UK (telephone: +44-2078827982, e-mail: josh.reiss@elec.qmul.ac.uk).

process have been thoroughly studied [4-8]. Besides time efficiency benefits, it was shown that learning from accelerated material can be at least equally as effective as the normal speed of presentation. There were further findings that students watching accelerated material stay more focused. At normal speech rates they "become bored and their attention begins to wander" [7], and learning processes benefit from acceleration of presentation as long as intelligibility can be maintained [8]. For entertainment applications, internet video streaming, digital video players and set-top devices can benefit greatly from an audio/video time stretching tool. Studies of digital video browsing [9] noted that one of the highest rated enhanced features was watching time compressed video.

II. Audio Time-scale Modification

Time-scale modification can be achieved in a number of ways in both the time and frequency domain. However, time domain approaches are typically not considered ideally suited to mixed audio content, which may include speech, polyphonic music and ambient noise. As such, the real-time time-scale modification technique proposed here is based on a set of modifications to the phase vocoder [10], a popular frequency domain approach to time-scaling. A comprehensive tutorial outlining the theory of the traditional phase vocoder is presented in [11] and a brief description is provided here.

The Fourier transform interpretation of the phase vocoder is mathematically equivalent to a short time Fourier transform (STFT) [12] which segments the analysed signal into overlapping frames which are separated by a certain 'hop size'. Within phase vocoder implementations, TSM is achieved by varying the analysis hop size R_a with respect to the resynthesis hopsize R_s such that the time scaling factor is calculated as $\alpha=R_s/R_a$. It follows then that $R_a>R_s$ will result in timescale compression (speed up), and $R_a<R_s$ will result in timescale expansion (slow down). Within the phase vocoder, analysis frames are 'remapped' along the time axis resulting in newly constructed synthesis frames, each with a modified phase spectrum, to ensure that the synthesis frames maintain phase coherence through time. Since the phase spectrum of each frame must be modified, the windowing function will also be affected. For this reason, a resynthesis window is necessary and a 75% overlap is recommended to avoid modulation at the output. This will result in the output having a constant gain factor of approximately 1.5 which can easily be compensated by multiplying all samples by the reciprocal of the gain factor. An overlap of 75% corresponds to a fixed synthesis hop size, R_s, of $N/4$ samples.

In order for the synthesis frames to overlap synchronously, the frame phases must be updated such that phase continuity is maintained between adjacent output frames. The standard method used to calculate suitable synthesis phases involves calculation of the instantaneous frequency of each bin in radians per sample. Having obtained the instantaneous frequency, it is possible to predict the expected phase of any component for a given synthesis hop size. Given that the frequency content of both music and speech is stationary only over short periods, phase estimates will decrease in accuracy as the hop sizes increase. The most accurate way to estimate phase for each component is by first calculating the principal argument of the heterodyned phase increment between adjacent analysis frames as defined in [10, 11]. The instantaneous frequency is then calculated in radians per sample. In order to calculate the phase spectrum for the new synthesis frame at the time scaled output, the instantaneous frequency is multiplied by the synthesis hop size R_s, and added to the resultant synthesis phases from the previous frame. This is known as phase propagation or phase updating. The newly modified phases along with the original magnitude spectrum are then used to reconstruct the audio frame.

Although, the time scaled output is horizontally phase coherent at this point, the timbral quality is often described as sounding *'phasey'* or '*distant'* and is generally not regarded as natural sounding. Particularly noticeable is how transients are affected by the phase vocoder. These artifacts can be attributed to the fact that the standard phase vocoder only attempts to achieve an optimal phase relationship between adjacent frames, known as horizontal phase coherence. However, the pursuit of horizontal phase coherence has a profoundly negative effect on vertical phase coherence, which describes the relationship between the phases of frequency components within a single frame. Maintaining vertical phase coherence is an important consideration in order to achieve natural sounding TSM.

The improved phase vocoder [13] explicitly attempts to identify sinusoidal frequency bins in FFT frames by a peak picking process within the magnitude spectrum. The phases of these truly sinusoidal peak frequency bins are then updated in the traditional manner, i.e., by maintaining horizontal phase coherence between corresponding peak frequency bins of successive frames. The non-sinusoidal frequency bins are then updated by maintaining the phase difference that existed between each bin and its closest peak/sinusoidal frequency bin. The process is known as peak locking.

III. Real-time Considerations For Dynamic Time-Scaling

When a fixed time-scale factor is applied to an entire audio signal both R_a and R_s remain fixed. In which case, the position in time of any analysis or synthesis frames can be defined as $t_a^u = uR_a$ and $t_s^u = uR_s$, respectively, where u is an incrementing integer representing a sequence of frames as in [10]. For real-time implementations, where the time-scale factor, α, may be varying dynamically due to user intervention, this definition will introduce distortions into the time-scaled output since the analysis hop is no longer fixed. The solution is to redefine $t_a^u = uR_a$ as $t_a^u = t_a^{u-1} + \alpha R_s$. This ensures that the current analysis frame position t_a^u, is always updated correctly. The position in time of the current analysis frame is always related to both the previous analysis frame and the current time scaling factor, α.

Although it is favourable to vary the analysis hop R_a and fix the synthesis hop R_s to achieve TSM, it can result in inaccurate

frequency estimation for time-scaling factors α < 1. When the signal is being sped up, the distance between analysis frames exceeds *N*/4. It becomes impossible to accurately predict the amount of phase unwrapping to be applied during the frequency estimation stage of the horizontal phase update procedure described in [10, 11], resulting in inaccurate synthesis phase estimates. In addition to this, when α is varied over time, the accuracy of the instantaneous frequency estimates also varies. This leads to momentary artefacts whenever the time scale factor, α, is changed. Effectively, the transitions between frames with different TSM factors are not perceptually smooth despite the windowed overlapping scheme. The solution to both of these problems is to ensure that the instantaneous frequency estimates are always derived using the phase differences between the current analysis frame and a frame one synthesis hop back from the position of the current analysis frame, $\angle X\left(t_a^u - R_s, \Omega_k\right)$. Although, an extra FFT and an extra buffer is required to obtain the phases of this frame, it guarantees that phase unwrapping errors will not be present and that the instantaneous frequency estimates will be consistent regardless of variation in α. The phase update equation [10, 11] is now redefined in (1).

$$\angle Y(t_s^u,\Omega_k)=\angle Y(t_s^{u-1},\Omega_k)+\angle X(t_a^u,\Omega_k)-\angle X(t_a^u-R_s,\Omega_k) \quad (1)$$

When vertical phase coherence is to be maintained, peak locking can be used, and only the sinusoidal or peak frequency bins are updated using (1), with all other bins updated as in [10, 11]. This method of phase updating removes the need to estimate the instantaneous frequency. However, for the case where pitch scale modification is required, calculation of instantaneous frequency is still necessary. Nonetheless, the 'hop-back' method described above is used to avoid phase unwrapping errors and to maintain smooth pitch and time scale transitions. This will be discussed in the next section.

A similar phase update procedure was proposed in [14] in which time-scale modification is achieved through the insertion and deletion of entire frames. Since the approach we propose here uses a variable analysis hop size, it has the advantage of maintaining better estimates of the magnitude spectrum, thereby greatly reducing the possibility of removing or repeating perceptually salient characteristics within the time-scaled signal.

IV. Real-time Pitch Shifting

The simplest method to shift the apparent pitch of a signal is by interpolating or decimating the time domain signal. The resulting signal, although pitch shifted, is also shortened or lengthened by the reciprocal of the interpolation/decimation factor β. A common technique used to shift the pitch and maintain duration is to pitch scale the signal using interpolation/decimation, and apply complimentary time scale modification to restore the original length of the signal. This is easily achieved in the offline context but becomes difficult to implement in a real-time context. If both pitch shifting and time scaling are required simultaneously, the problem becomes more difficult since time scaling is required for 2 alternate operations (pitch and time scaling) within the same frame. When the signal is both time scaled and interpolated for any time scaling factor α and pitch shifting factor β, the required compensatory time scale factor such that the resultant signal is both the required pitch and length [15], is simply $\alpha\beta$.

In a real-time context the pitch and time scaling must be carried out within a single frame interval (in this implementation 23ms). Two issues arise. First, the computational requirements are directly related to the product of α and β, since each frame must now be time-scaled internally to compensate for pitch shifting. This makes real-time operation unfeasible for large products $\alpha\beta$. Second, the length of the resultant frame is no longer fixed. An additional buffer must be used in order to handle the overflow if the resultant frame exceeds *N* (analysis frame size) samples. If $\alpha\beta$<1, the resultant frame will be smaller than the required *N* samples. In this case, more input frames need to be processed until there are sufficient samples to generate an output frame. These issues can make the output unpredictable, added to which the solutions are computationally intensive.

Here we present a novel method for real-time pitch shifting which resolves the problematic issues raised above. The computational requirements are not dependent on α and β and the method guarantees that a fixed frame length can be generated independent of the time and pitch scale factors used. No inter-frame time scaling and no additional buffers are required. The pitch shifting is performed using linear re-sampling in the time domain, and phase vocoder theory is then applied using a modified phase update equation which incorporates the pitch scaling factor β. In order to generate a pitch shifted frame of known length, we interpolate or decimate the input time domain signal over the range t_a^u to $t_a^u + N\beta$, where *N* is a fixed analysis frame size chosen to ensure adequate frequency resolution. This results in a time domain frame of length *N* which has been generated by interpolating or decimating $N\beta$ samples by the pitch scaling factor β. Figure 1 illustrates this procedure. This frame now constitutes an analysis frame which can have arbitrary time scaling applied using the phase update equations presented below.

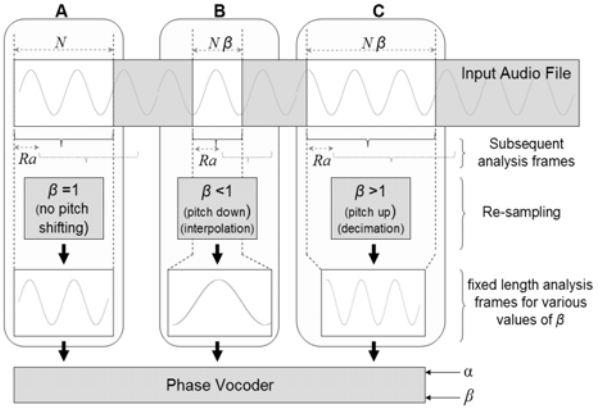

Figure 1. The real-time re-sampling method used for obtaining fixed length pitch shifted frames. A illustrates no pitch change, B pitching down and C pitching up.

The goal is to estimate the phase propagation required to allow successive interpolated frames to be updated such that the pitch shifted and time scaled output is horizontally phase coherent. Recall (1), which was introduced as a preferred method to ensure reliably wrapped phase difference estimates. This was achieved by using an extra FFT to estimate the phases of the frame exactly one synthesis hop back from the current analysis frame, thereby allowing the phase differences to be estimated over a known fixed interval equal to R_s. The 'apparent' analysis hop is now equal to the synthesis hop, but the actual value of R_a is still variable. In order to estimate suitable synthesis phases for pitch shifted frames, the instantaneous frequency must be calculated as follows. A new method to calculate the heterodyned phase increment for pitch shifted frames is given by (2), where the interpolation factor, β, is now included in the equation.

$$\Delta_p\Phi_k^u = \angle X(t_a^u,\Omega_k) - \angle X(t_a^u - Rs,\Omega_k) - R_s\Omega_k/\beta \quad (2)$$

where $\angle X(t_a^u,\Omega_k)$ and $\angle X(t_a^u - Rs,\Omega_k)$ represent the phases of the current analysis frame and an analysis frame exactly one synthesis hop back from the current value of t_a^u. The resulting term, $\Delta_p\Phi_k^u$, is then the principle argument of the heterodyned phase increment of the pitch shifted frame such that it is in the range -π to π. Since the frames have been interpolated or decimated (resulting in frequency shifts) they will no longer exhibit the expected phase derivatives over a given hop, R_s. To calculate the correct phase increment, the hop must also be multiplied by the reciprocal of the pitch scaling factor, β. The instantaneous frequency in radians per sample of the pitch shifted frame is given by (3).

$$\hat{\omega}_k(t_a^u) = \Omega_k + \beta\Delta_p\Phi_k^u / R_s \quad (3)$$

As opposed to the standard method [10, 11], we divide the phase deviation by R_s instead of R_a, because the method used to calculate phase difference in (3) uses two frames separated by a fixed distance, R_s. The standard phase update equation [10, 11] can now be used, and peak locking can be applied as discussed previously. The advantages of using (1) for phase updating have already been incorporated in (2) above. We now have modified phase vocoder equations which allow real-time pitch shifting and time stretching simultaneously. A key advantage of using this method for pitch shifting is that compensatory time scaling is not required. Instead, the pitch scaling factor is incorporated in the phase update equations. This guarantees that the computational load remains fixed and predictable for any combination of time and pitch scaling factors.

V. Real-time Transient Preservation

Although peak locking contributes to maintaining the timbral quality of transients during TSM, transients should not be time-scaled if a naturally sounding output is required. An off-line solution was proposed in [16]. The approach taken here is to identify transients automatically in real-time. Upon detection of a transient, the time scale factor α is returned to '1' (no scaling), and the analysis phases are mapped directly to the synthesis phases (phase locking) for the duration of the transient. When the transient has passed, the time scale factor is automatically reset to the α value prior to the transient. Transients represent an ideal place to lock the phases since any discontinuities introduced to the time scaled signal will be masked by the transient itself.

In order to identify an analysis frame as a transient [17], the log difference of each frequency component between consecutive frames is calculated as in (4). This measure effectively tells us how rapidly the spectrogram is fluctuating.

$$X_f(t_a^u,k) = 20\log_{10}\frac{\left|X(t_a^u,k)\right|}{\left|X(t_a^u - R_s,k)\right|}, \quad 1 \le k \le N/2 \quad (4)$$

where $X_f(t_a^u,k)$ is the log energy difference between frames separated by R_s, and t_a^u is the current analysis frame instant. In order to detect the presence of a transient we define a measure given in (5).

$$Pe\left(t_a^u\right) = \sum_{k=1}^{N/2}\begin{cases} P\left(t_a^u,k\right)=1 & if \;\; X_f(t_a^u,k) > T_1 \\ P\left(t_a^u,k\right)=0 & otherwise \end{cases} \quad (5)$$

where, T_1 is a threshold which signifies the rise in energy, measured in dB, which must be detected within a frequency channel before it is deemed to belong to an onset. In order for the frame to be declared a transient, $Pe(t_a^u)$ must exceed a second threshold T_2. In practice we have found that T_1=6dB and T_2=3N/8 give satisfactory results for most popular music. Thus, a transient is detected at frame t_a^u, if at least 75% of the bins in the log difference spectrogram, equation (4), exceed a value of 6dB. Note that using this measure, the energy present in the signal is not the defining factor of the transient. Instead, we assign the transient probability, $Pe(t_a^u)$, using a measure of how "broadband" or percussive the onset is [17]. This is based on the number of bins exhibiting a positive first derivative as described by equation (5). Figure 2 shows the effectiveness of this approach. Despite the fact that the signal itself has little dynamic range, the feature detector is rarely prone to false detections which makes it ideal for transient detection in time scaling. Furthermore, it can easily be implemented within the current framework since the only requirement is that the current and previous frame magnitudes are available.

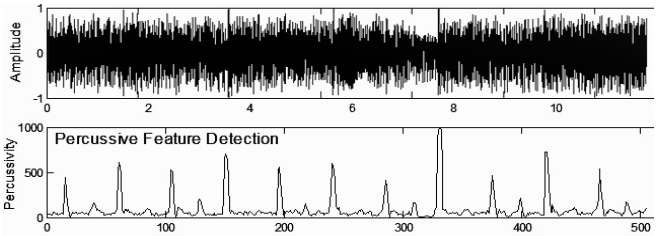

Figure 2. A highly dynamically compressed signal containing rock music is depicted in the top plot. The bottom plot shows the output of the percussive onset detector.

Upon detecting a transient, the time scale factor, α, is automatically returned to '1', inhibiting TSM momentarily.

We term this method 'transient hopping'. In addition the frame phases are locked and the frame is mapped directly to the output. This mechanism preserves the transient and ensures that it is reproduced unaffected at the output. Since we use 75% overlap, R_s = 1024 for analysis frame length 4096, a short transient will exist in 4 consecutive frames. In order to preserve the transient correctly, the TSM factor, α, must remain at a value of '1' until all overlapping frames have passed the transient. Since the local time scale factor is reduced, a time scale compensation factor is applied after each transient. Eq. (6) describes this action:

$$\alpha^T = \begin{cases} 1 & if\ u - u^T < 4 \\ \dfrac{\alpha m - \alpha}{m - \alpha} & if\ 4 \le u - u^T < N_F + 4 \\ \alpha & otherwise \end{cases} \quad (6)$$

where α is the global time scale factor, α^T is the TSM factor to be applied during the frames preceding the transient, and where m is the maximum desired TSM factor and m must be strictly greater than 1. The number of frames, N_F, in which the time scale compensation factor must be applied after the transient, is dependent on the maximum timescale factor, such that $N_F = 4m\text{-}4$. Using a larger number of frames to compensate for the transient has the advantage that smaller TSM factors may be distributed over a longer time period, thus reducing signal distortion due to excessive timescale factors.

Figure 3 illustrates how the time scaling factor is varied before and after the transient in order to both preserve the transient and to maintain a constant global time scale factor.

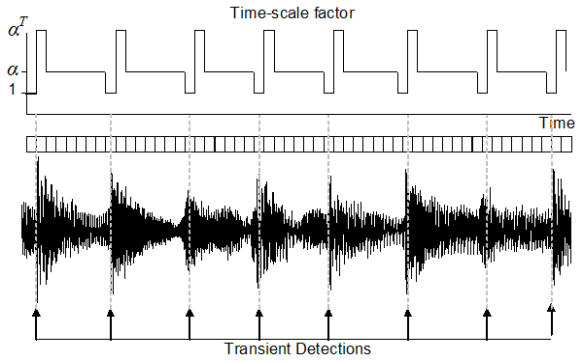

Figure 3. Time scale factor as a function of transient detection

VI. Buffer Schemes

One of the key issues in a real-time implementation of TSM is the choice of buffer scheme and for completeness sake we suggest a suitable scheme here. In offline processing, the entire signal is overlapped and concatenated before playback. However, in a real-time environment, a constant stream of processed audio must be outputted and consecutive output frames must be continuous. In order for seamless concatenation, the boundaries of each output frame must be at the constant gain associated with the overlap factor in order to avoid modulation. The method presented below addresses this concern. For reasons discussed in previous sections, a 75% overlap is recommended. This effectively means that at any one time instant, 4 analysis frames are actively contributing to the current output frame.

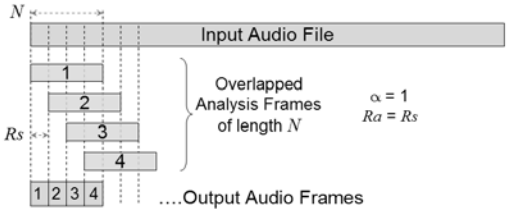

Figure 4. The relationship between input and output frames for α=1.

In Figure 4, the audio to be processed is divided into overlapping frames of length N. In order to output a processed frame, 4 full frames would need to be processed and overlapped. This leads to considerable latency from the time a parameter change is affected to the time when its effects are audible at the output. However, given that the synthesis hop size is fixed at R_s=N/4, we can load and process a single frame of length N, output ¼ of the frame, and retain the rest in a buffer to overlap with audio in successive output frames. To do this, a buffer of length N is required in which the current processed frame (with synthesis window applied) is placed. Three additional buffers of length 3N/4, N/2 and N/4 will also be required to store remaining segments from the 3 previously processed frames. Each output frame of length N/4 is then generated by summing samples from each of these 4 buffers. Figure 5 shows how the buffer scheme works. On each iteration u, a full frame, F^u, of length N is processed and placed in buffer 1. The remaining samples from the 3 previous frames occupy buffers 2, 3 and 4. The required output frame of length N/4, S^u, is generated as defined in (7).

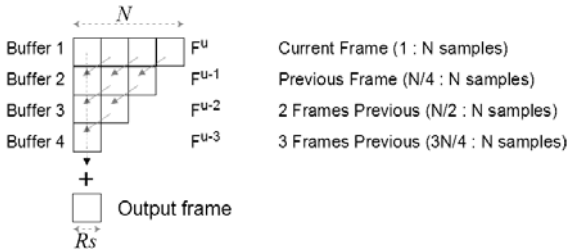

Figure 5. Real-time output buffer scheme using a 75% overlap. The gray arrows indicate how each segment of each buffer is shifted after the output frame has been generated.

$$S^u(n) = F^u(n) + F^{u-1}(n+N/4) + F^{u-2}(n+N/2) + F^{u-3}(n+3N/4) \quad (7)$$
$$\forall n \qquad 1 \le n \le N/4$$

From (7), it can be seen that the output frame, $S^u(n)$, is generated by summing the first N/4 samples form each buffer. Once the output frame has been generated and outputted, the first N/4 samples in each buffer can be discarded. The data in all buffers must now be shifted in order to prepare for the next iteration. The gray arrows in Figure 5 illustrate how each segment of each buffer is shifted in order to accommodate a newly processed frame in the next iteration. The order in which the buffers are shifted is vital. Buffer 4 is filled with the remaining N/4 samples from buffer 3, buffer 3 is then filled

with the remaining $N/2$ samples from buffer 2, and finally buffer 2 is filled with the remaining $3N/4$ samples from buffer 1. Buffer 1 is now empty and ready to receive the next processed frame of length N. The result of this scheme, is that ¼ of a processed frame will be outputted at time intervals of R_s, which is equal to $N/4$ samples. Using the suggested frame size of 4096 samples, the output will be updated every 1024 samples which is approximately equal to 23.2 milliseconds. The audio will be processed with newly updated parameters every 23.2 milliseconds, but the latency will be larger than this and depends on the time required to access and write to hardware buffers in the audio interface. In general however, it is possible to achieve latencies < 40ms.

VII. Synchronisation with the Host Application

The requirement to synchronise independent time and pitch scaling with video and screen updates adds additional complexity. To maintain multimedia synchronisation, the time scaling process should control the master clock within an application. In this section, we present a real-time media synchronisation framework which has made this possible

Previous sections have described in detail the audio processing blocks required to achieve real-time time and pitch scaling simultaneously. Figure 6 shows how the overall system is configured.

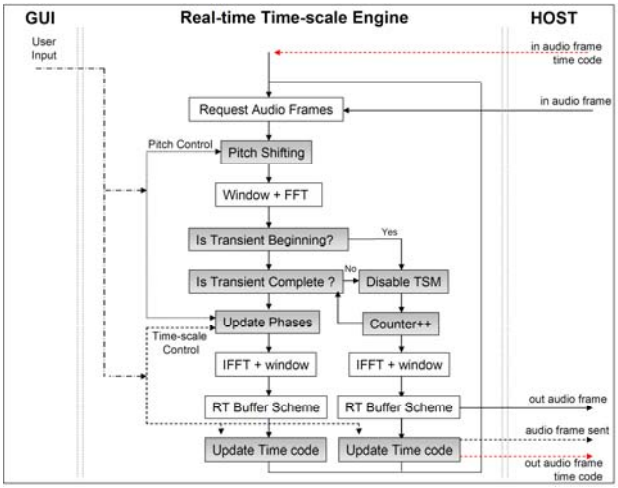

Figure 6. Overview of clocking between time/pitch scale modification and host application.

Firstly, it is important to note that, in order to allow time scale modification to be carried out in real-time whilst maintaining synchronisation with other media such as video or screen updates, e.g., the audio locators, it is necessary to pass full control of the host clock to the time scaling algorithm. This is because time scaling by its very nature involves manipulation of the time base of the audio. As described previously, the time increment between frames is purely dependent on the choice of time scale factor. Furthermore, if we wish to continuously vary the time-scale factor, the time line becomes non linear at transition points. Essentially, the time scale algorithm must be able to request *any* audio frame, starting at *any* sample point within the audio stream.

With this in mind, the first stage involves loading an audio frame defined by the time scale algorithm itself. Immediately following this, the first stage of pitch shifting is achieved by interpolating or decimating the input waveform by the pitch scaling factor. Regardless of time or pitch scale factor, one full frame is always populated on every iteration. For example, using a pitch scale factor of '2', $2N$ samples will be interpolated to produce an N sample frame where N is the frame size. If the frame is identified as a transient, no further processing is applied, and time scaling is suspended for 4 frames (due to 75% overlap). The frames around a transient are reproduced at the output identical to that of the input and the audio clock is updated as normal. If no transient has been detected, the phases are updated according to the modified phase update equations. Pitch shifting is only completed at this stage since the phase update procedure needs to include the interpolation factor. Following this, the processed audio frame is reproduced and re-windowed. The audio clock is then updated, and the frame incremented by a varying factor depending on the user input (i.e., TSM factor). In order to produce a continuous stream of audio, the buffer scheme described above is used.

Regardless of what processing is carried out by the time-scaling algorithm, it is solely responsible for updating the host clock. The host then uses this information to update screen components which depend on audio playback position. Thus, all screen components, processes and visualisations are synchronised with the audio clock which is controlled by the time-scale modification algorithm.

VIII. Video Synchronisation

Combined audio/visual artefacts that can be introduced due to loss of synchronisation are often the most perceptually undesirable. Failure to keep audio and video streams synchronised, known as 'lip sync errors,' result in audio events occurring before or after the associated video frames. When audio advances video by 20ms or when audio lags video by 40ms, it becomes detectible. Errors of +40ms and -160ms are "subjectively annoying" as reported by the International Telecommunications Union (ITU) in 1993[18]. Further research reported in ITU-R BT1359-1 [19], showed reliable detection of 45ms audio leading and 125ms audio lagging, while the acceptability region is even wider. The ITU recommends that the difference between audio and video should be no less than -90ms and no more then +185ms. In reality, this range is probably too wide for acceptable performance. For example, in video footage of musical instruments being played, key strokes or string plucks are more precise than lip movement during speech, so the synchronisation thresholds need to be reduced. In addition, when a video has been stretched it can be easier to analyse and therefore synchronisation errors become more perceivable.

In this section, three approaches to the preservation of audio/video synchronisation in time scaling applications will be presented. Insertion and deletion of frames is necessary when the frame rate is dictated by the playback device. Television standards such as PAL/SECAM and NTSC use

standardised refresh rates and hence the output of a time stretching module must maintain a corresponding frame rate. However, many software implementations of video players, including mplayer, VLC player and others, allow for change of the video rate once the compressed video is unpacked. Screen refresh rate of modern equipment is in the range of 100-200Hz, so variations in the frame rate can be introduced by choosing when a particular video frame will be shown on the display device. Hence, less noticeable artefacts and smoother picture transition can be obtained when variable frame rate, the second method, is applied. The third method, Adaptive Video Refresh Rate (AVRR), relies on the precision of the audio clock. Synchronisation is maintained by ensuring that the video time code remains locked to the audio time code within an allowable threshold.

Video time stretching for conventional broadcast uses insertion and deletion of frames to maintain synchronisation. When speeding up the video, some frames need to be dropped, whilst when slowing down some need to be duplicated. When frames are duplicated or dropped, maximal synchronisation error is half of a video frame length, since we round to the closest frame. Hence, if the frame rate is 25 fps, maximal error will be 20 ms. This error range (-20ms to +20ms) meets ITU recommendations for lip sync error to be undetectable. However it may not be good enough for more demanding applications such as time stretching of video, when precise movements are slowed and become easier to analyse. In addition, frame duplication can cause jerkiness to be perceived in the video of slow steady movements.

Changing the video frame rate by the scaling factor will generally give a smoother image since frames are equally spaced in time. The additional advantage is that no frames are dropped when speeding up. Ideally, timing for a new frame is easy to calculate by advancing the previous frame time by the new frame rate interval. However, due to the fact that timing precision is influenced by factors such as temperature and humidity, simply setting-up the next frame to display a given period after the previous frame without comparing it to a master clock can cause long term synchronisation errors.

The AVRR method refreshes the display with a new frame when the video time code is equal to (or within a threshold of) the original time code of the audio frame being outputted. The refresh rate is adaptive since the period between two frames adapts to the audio clock. Ideally, it should be equal to the reciprocal of the scaled frame rate, but will oscillate around that value. We define here two time-lines; one is the media player's actual time-line and the other is the original media time-line. It is crucial for this method to calculate precisely the time on the media time-line of the audio sample currently being played. This time value is then compared with the original time code associated with non-time scaled video frames and the display is refreshed with this frame when the video frame time code is smaller than or equal to the time of the audio sample that is currently being outputted. To minimise loss of synchronisation due to computationally intensive processing, the decoding algorithm needs to be efficient and implemented in a separate high priority thread.

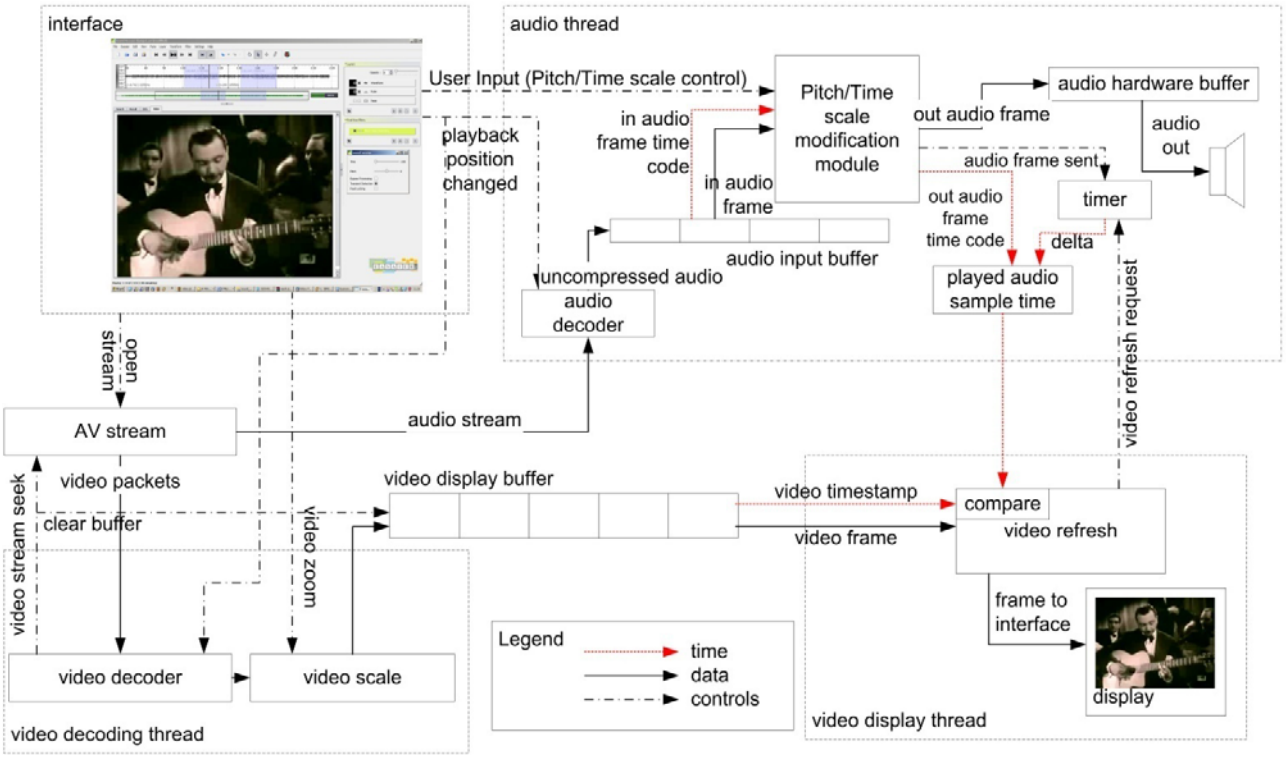

Figure 7. Video time scaling implementation.

The video-synchronised time stretching algorithm described above was implemented as presented in Figure 7, and intended for a demanding application requiring fast access to audio frames while other intensive processing tasks are performed. Here, the audio stream is first uncompressed and stored locally in an audio input buffer. Unlike audio however, uncompressed video would require an unacceptably large local buffer, so video packets are accessed directly from the compressed stream.

Since video decoding is done on-line, particular consideration was given to its implementation. Higher time compression rates will demand that video frames be decoded and scaled much faster than usual. Hence, the video decoding is carried out together with video zooming in a separate high priority thread. The video decoding thread receives two control inputs from the user interface. Video zoom factors, changeable from the interface, are sent directly to the video scaler, which scales a frame according to a zoom factor and sends it to the video display buffer. Change of playback position is sent to the decoder and it instructs the decoder to seek the stream and also to erase any previously decoded frames from video display buffer.

The time-stretching factor is sent to the audio processing engine in order to change the analysis hop size, and the audio output frame timestamp is calculated accordingly. However, this timestamp is not sufficient for proper A/V synchronisation, since it represents the time when the audio frame is sent to the audio hardware buffer. For example, if an audio frame is 1024 samples and the sample rate is 44100 Hz, the time resolution will be 23.2 ms. For the normal playback speed, this may be sufficient, but in the case of doubling the playback speed the time span between two audio sample points on the media timeline becomes 46.4 ms. Hence, some measure of fullness of the audio hardware buffer needs to be introduced for precise timing of outputted audio samples. The fullness of the hardware audio buffer is hardware dependent and measuring it is often a complex task, so we propose to find approximate timing of the audio sample by measuring the time difference (Δt) between the moment the audio frame is sent to the hardware buffer and the current time. This value is then added to the timestamp of the audio frame that was sent to the audio buffer (T_{audio}), and is then compared with the video frame timestamp (T_{video}). The display is refreshed with this frame when the video frame time code is smaller than or equal to the calculated audio time:

$$T_{video} \le T_{audio} + \Delta t \tag{8}$$

Another issue is timer precision for measuring Δt. In Windows OS, the maximal precision that can be achieved with the standard timer is 15ms, which is hardly enough for a synchronisation application. Hence, Δt is determined by measuring CPU counts from the moment the frame is sent to the hardware buffer and then dividing by the CPU count frequency. Since Δt gives a value related to the real playback time-line, it is transposed to the media time line by dividing it by the time-stretching factor α:

$$\Delta t = \frac{1}{\alpha} \cdot \frac{CNT_{cpu}}{f_{cpu}} \tag{9}$$

However, both variable frame rate and adaptive video refresh rate have the potential disadvantage that at higher time scale factors, since more frames are displayed per second, frames need to be decoded much faster. Synchronisation can be lost if a frame is not decoded within a frame interval, so a preferred solution is to combine AVRR with frame dropping when loss of synchronisation occurs. In our implementation, whenever the video lag exceeds 20 ms, the application instructs the decoder not to decode the following frame, and returns to full decoding when the lag returns to under 10 ms.

IX. AUDIO QUALITY EVALUATION

Since the focus of this research is concerned with the real-time implementation of a synchronised video/audio and multimedia time and pitch scale modification algorithm, the evaluation of the audio time-scale algorithm presented here is not intended to be comprehensive. Instead, to ensure that this real-time implementation has not resulted in a compromise to the audio quality of the algorithm, a series of subjective listening tests were carried out in order to ensure that the TSM algorithm is as least as good as that described in [13]. The transient detection has not been used in these comparison tests since [13] does not employ transient detection.

In total, 10 subjects undertook a series of 20 tests[1] each, totalling 200 individual tests. The tests used included slowing and speeding of audio as well as pitch shifting in both directions by a range of factors. Both time and pitch scale factors ranged from 0.75 to 1.5. A range of signals including solo and ensemble music from a range of genres and male and female speech segments sampled at 16 bit, 44.1 kHz comprise the test suite. Each listener was presented with an unprocessed reference signal and two alternative processed signals. The same processing parameters and frame sizes are used in each algorithm. The order in which the algorithms are presented was randomised.

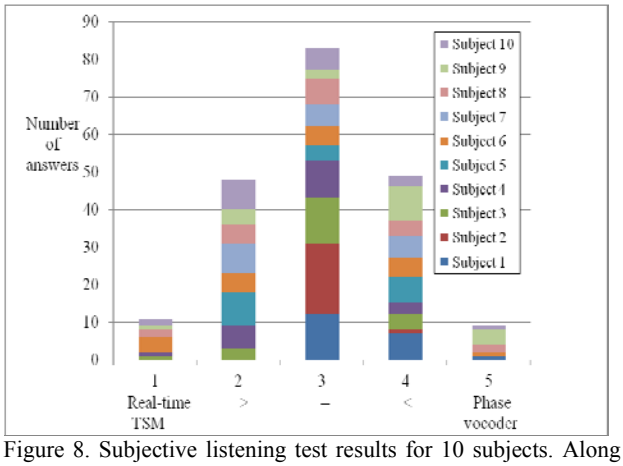

Figure 8. Subjective listening test results for 10 subjects. Along the horizontal axis, 1 indicates a predominant preference for real-time TSM whereas 5 indicates predominant preference for the improved phase vocoder [13].

[1] http://www.audioresearchgroup.com/downloads/tsmtests.zip

The results are presented in Figure 8, where results for each subject are given from 1 to 5, where 1 indicates predominant preference for real-time TSM, 3 indicates no preference, and 5 indicates predominant preference for the improved phase vocoder. The subjective listening tests indicate that the overall trend is such that the algorithms are perceived to perform equally well. The average value over all 200 tests was 2.985, very close to no preference, with a relatively low standard deviation of 0.94. Subjects who were predisposed to distinctly choosing 1 algorithm over the other tended to choose each algorithm a similar number of times indicating equivalence of the algorithms. Many subjects reported that the algorithms sounded very similar but felt compelled to make explicit decisions regardless. The data is skewed slightly in favour of the real-time TSM algorithm, but it is likely that a greater number of test subjects would introduce greater balance in the data. Some differences between the algorithms which may account for this include the fact that the real-time TSM algorithm does not perform peak locking above 10 KHz due to the fact that peak locking is intended to maintain the phase relationship between the peak and lobes of sinusoidal components. Significant acoustic energy above 10 KHz is often stochastic and attributed to transients, noise and ambience. Peak locking above 10 KHz forces non-sinusoidal components into a state of unnatural phase coherence which can sound objectionable to subjects with acute hearing in the upper frequency range.

Theoretically, the pitch shifting quality in [13] should outperform that of the real-time algorithm but subjective tests have shown that the differences are largely imperceptible for moderate time scaling factors (in the region of 0.75 to 1.5) although the real-time algorithm can become noticeably more objectionable when opposing time and pitch scale factors are used simultaneously (i.e. slow down and pitch up simultaneously). This is due to the efficient pitch shifting technique used to achieve frame synchronous pitch shifting.

X. A/V SYNCHRONISATION EVALUATION

To measure the quality of the A/V synchronisation algorithm, we compared it with integration of our time-stretching into the FFmpeg (v0.4, ffmpeg.org/ffplay-doc.html) platform and with the MPlayer implementation (v1.0rc2, www.mplayerhq.hu/) in LinuxOS. FFplay is a well known efficient open source application for video encoding, and MPlayer is a robust, open source video player based on ffmpeg libraries. One of the many features of MPlayer is the possibility to change playback speed, but without independent pitch-shifting. Nevertheless, this feature, robust implementation and the possibility to extract A/V synchronisation information make MPlayer useful for evaluation and comparison with our algorithm. For A/V synchronisation, FFplay uses duplicating and dropping video frames whereas MPlayer uses a variable frame rate.

We compared video players on the “Casino Royale” trailer sequence coded in MPEG1 format with video frame dimension 640x352 at 23.97 frames per second and an audio sample rate of 44100 Hz. The video frame lag with respect to audio is presented for 100 video frames from the middle of the sequence in the case of playing the video at half of the original speed (Figure 9) and with double the original speed (Figure 10). It can be seen that our adaptive video refresh rate algorithm clearly outperforms the other two, because of the precise matching of the video timestamp to the audio clock. The video lag of the AVRR time-stretching algorithm is also well below the ITU lip sync error recommendation with maximal video lag being 14 ms and maximal video advance being 13 ms in the case of doubled playback speed. Moreover, the standard deviation of video lag is 3.328 ms, showing stability of this solution.

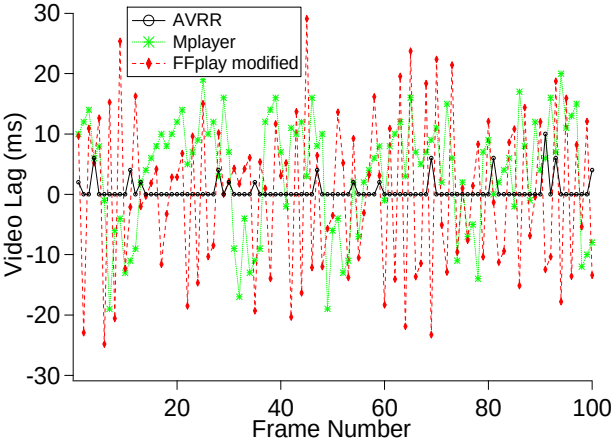

Figure 9. Comparison of video lag for three video player implementations when playback speed is half of original.

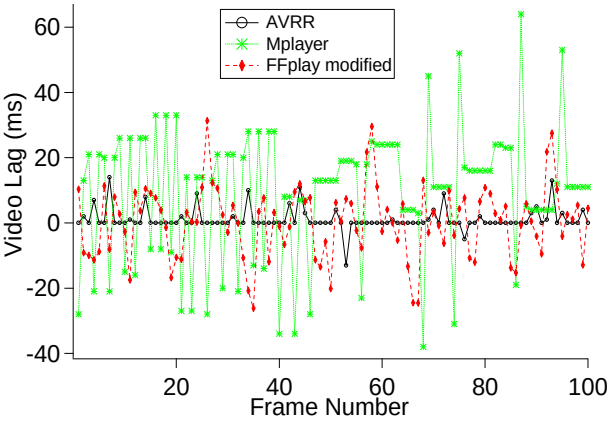

Figure 10. Video lag when playback speed is doubled.

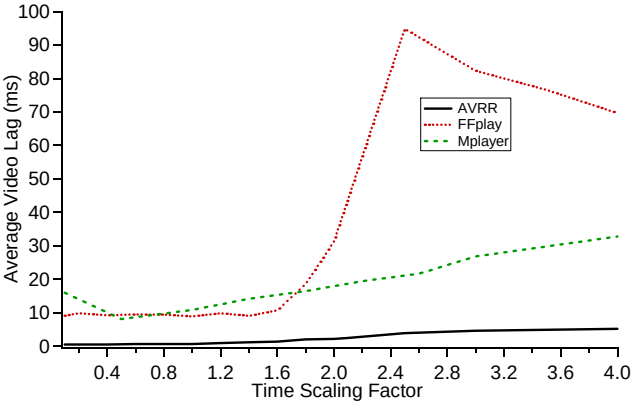

Figure 11. Average video lag as a function of time scaling factor.

Figure 11 depicts the average video lag as a function of the

time scaling factor for the three video synchronisation techniques. FFplay was modified to ensure that it would not decode dropped frames, otherwise its performance would be significantly worse. However, it still shows notable degradation in performance as the time scaling factor increases beyond 2 and video frame decoding becomes significantly slower than the time to process a time scaled audio frame. MPlayer maintains suitable performance as time scale increases, though it does not adapt the variable refresh rate to the precise audio time codes. The AVRR method maintains strong synchronisation over the entire range of time scaling factors. Only at time scaling factors beyond 3.5 does the AVRR occasionally lose synchronisation, and opts not to decode a frame.

XI. Conclusions

A framework for real-time independent video time scaling and pitch shifting was presented. Careful consideration was given to the problems which arise in a real-time context and novel solutions to these issues have been provided. It was shown how time-scale changes can be achieved in real-time with almost imperceptible latency and no transitional artefacts. The approach is based on a modified phase vocoder with optional phase locking and an integrated transient detector which enables high quality transient preservation in real-time.

The framework presented is the basis for the developments of applications which allow for a seamless real-time transition between continually varying, independent video time-scale and pitch-scale parameters. A novel solution for audio/visual synchronisation called adaptive video refresh rate has also been developed. Due to the fact that synchronisation errors in the foreseen applications will be easier to detect, special focus was given to minimizing video lags and advances, resulting in an algorithm that significantly outperforms existing algorithms.

References

[1] M. C. Yuang, S. T. Liang, and Y. G. Chen, "Dynamic video playout smoothing method for multimedia applications," *Multimedia Tools and Applications*, vol. 6, pp. 47-59, 1998.

[2] M. Kalman, E. Steinbach, and B. Girod, "Adaptive Media Playout for Low Delay Video Streaming over Error-Prone Channels," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 14, pp. 841-851, 2004.

[3] Y. J. Liang, N. Färber, and B. Girod, "Adaptive playout scheduling using time-scale modification in packet voice communication," presented at International Conference on Acoustics, Speech, and Signal Processing (ICASSP), pp. 1445-1448, Salt Lake City, 2001.

[4] P. LaBarbera and J. MacLachlan, "Time-Compressed Speech in Radio Advertising," *Journal of Marketing*, vol. 43, pp. 30-36, 1979.

[5] C. Landone, J. Harrop, and J. D. Reiss, "Enabling Access to Sound Archives through Integration, Enrichment and Retrieval: The EASAIER Project," presented at 8th International Conference on Music Information Retrieval (ISMIR), Vienna, 2007.

[6] C. Duffy, "A case study of networked sound resources for education in traditional music: the HOTBED project," presented at Integration of Music in Multimedia Applications, Barcelona, Spain, 2004.

[7] J. S. Olson, "A Study of the relative effectiveness of verbal and visual augmentation of rate-modified speech in the presentation of technical material," in *Annual Convention of the Association for Educational Communications and Technology (AECT)*. Anaheim, CA, 1985.

[8] K. Harrigan, "The SPECIAL system: Searching time-compressed digital video lectures," *Journal of Research on Computing in Education*, vol. 33, pp. 77-86, 2000.

[9] F. C. Li, A. Gupta, E. Sanocki, L. He, and Y. Rui, "Browsing digital video," presented at ACM CHI, Hague, Netherlands, 2000.

[10] J. L. Flanagan, D. I. S. Meinhart, R. M. Golden, and M. M. Sondhi, "Phase vocoder," *The Journal of the Acoustical Society of America*, vol. 38, pp. 939, 1965.

[11] M. Dolson, "The phase vocoder: A tutorial " *Computer Music Journal*, vol. 10, pp. 14-27, 1986.

[12] M. Portnoff, "Implementation of the digital phase vocoder using the fast Fourier transform," *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. 24, pp. 243- 248, 1976.

[13] J. Laroche and M. Dolson, "Improved phase vocoder timescale modification of audio," *IEEE Trans. Speech and Audio Processing*, vol. 7, pp. 323 -332, 1999.

[14] J. Bonada, "Automatic technique in frequency domain for near-lossless time-scale modification of audio," presented at International Computer Music Conference, pp. 396-399, Berlin, Germany, 2000.

[15] J. Laroche, "Autocorrelation method for high quality time/pitch scaling," presented at IEEE WASPAA, pp. 131-134, Mohonk, NY, 1993.

[16] C. Duxbury, M. Davies, and M. Sandler, "Improved time-scaling of musical audio using phase locking at transients," presented at 112th AES Convention, pp. 1-5, Munich, Germany, May 10-13, 2002.

[17] D. Barry, D. FitzGerald, and E. Coyle, "Drum Source Separation using Percussive Feature Detection and Spectral Modulation," presented at IEE Irish Signals and Systems Conference, pp. 13-17, Dublin, Ireland, 2005.

[18] International Telecommunication Union Document 11A/47-E, 13 October 1993.

[19] "Relative Timing of Sound and Vision for Broadcasting. Recommendation," International Telecommunication Union ITU-R BT. 1359-1, 1998.