NTIA Technical Memorandum TM-09-456

A No Reference (NR) and Reduced Reference (RR) Metric for Detecting Dropped Video Frames

Stephen Wolf

technical memorandum

U.S. DEPARTMENT OF COMMERCE · National Telecommunications and Information Administration

NTIA Technical Memorandum TM-09-456

A No Reference (NR) and Reduced Reference (RR) Metric for Detecting Dropped Video Frames

Stephen Wolf

U.S. DEPARTMENT OF COMMERCE
Carlos M. Gutierrez, Secretary

Meredith A. Baker, Acting Assistant Secretary
for Communications and Information

October 2008

DISCLAIMER

Certain commercial software are identified in this report to specify adequately the technical aspects of the reported results. In no case does such identification imply recommendation or endorsement by the National Telecommunications and Information Administration (NTIA), nor does it imply that the software identified is necessarily the best available for the particular application or use.

This document contains software developed by NTIA. **NTIA does not make any warranty of any kind, express, implied or statutory, including, without limitation, the implied warranty of merchantability, fitness for a particular purpose, non-infringement and data accuracy.** NTIA does not warrant or make any representations regarding the use of the software or the results thereof, including but not limited to the correctness, accuracy, reliability or usefulness of the software or the results. You can use, copy, modify, and redistribute the NTIA-developed software upon your acceptance of these terms and conditions and upon your express agreement to provide appropriate acknowledgments of NTIA's ownership of and development of the software by keeping this exact text present in any copied or derivative works.

CONTENTS

Page

A NO REFERENCE (NR) AND REDUCED REFERENCE (RR) METRIC FOR DETECTING DROPPED VIDEO FRAMES

Stephen Wolf [1]

Digital video transmission systems consisting of a video encoder, a digital transmission method (e.g., Internet Protocol – IP), and a video decoder can produce pauses in the video presentation that result from dropped or repeated video frames. For example, a common response of a video decoder to dropped IP packets is to momentarily freeze the video by repeating the last good video frame. This document presents a No Reference (NR) metric and a Reduced Reference (RR) metric for detecting these dropped video frames. These metrics may have application for in-service video quality monitoring.

Key words: dropped; frames; metrics; No Reference (NR); Reduced Reference (RR); video

1. INTRODUCTION

Digital video transmission systems consisting of a video encoder, a digital transmission method (e.g., Internet Protocol – IP), and a video decoder can produce pauses in the video presentation that result from dropped or repeated video frames. There are two primary reasons for this behavior. The first reason is that the video encoder may decide to reduce the video frame transmission rate in order to save bits. For example, an original video stream with a frame rate of 30 frames per second (fps) may be reduced to 15 fps by dropping every other video frame. The second reason is that the video decoder may decide to freeze the last good video frame when errors such as IP packet loss are detected. This is a simple error concealment algorithm that is used by many video decoders.

This document presents a No Reference (NR) metric and a Reduced Reference (RR) metric for detecting dropped video frames. An NR metric only requires access to the destination video stream to make a measurement (i.e., no access to the source video stream). An RR metric requires access to both the source and destination video streams, and a method to communicate low bandwidth reference information between the source and destination ends. NR and RR metrics are both useful for in-service quality monitoring applications.

The NR and RR metrics in this document are derived by examining the behavior of the motion energy in the video stream, which is computed from simple frame differences. Thus, these metrics should be easily implemented in real time by modern signal processing components.

---

[1] The author is with the Institute for Telecommunication Sciences, National Telecommunications and Information Administration, U.S. Department of Commerce, 325 Broadway, Boulder, CO 80305.

2. ALGORITHM DESCRIPTION

The NR algorithm for detecting dropped video frames was developed using sequences of captured video frames (often called video clips) that ranged from 5 to 10 seconds in length. The behavior of the algorithm for shorter or longer video sequences should be analyzed before being applied. The NR algorithm will estimate the number of dropped video frames in the video clip and their temporal locations. The RR version of this algorithm is obtained by applying the NR algorithm to both the source and destination video streams and then comparing the number of dropped video frames in the source and destination video clips. The RR version of the algorithm can thus correct for mistakes that might be due to source content (e.g., still or very low motion video scenes, slow motion video where frames are repeated). However, the price for this increased robustness is the additional computational complexity of applying the algorithm to the source video, the need to temporally synchronize the source video clip with the destination video clip, and the need to communicate the reference information between the source and destination ends.

This section provides a step by step description of the NR algorithm as applied to one video clip. The RR version of the algorithm requires an optional extra step, which is also described. The NR algorithm only utilizes the luminance images of the video clip (e.g., the Y channel in an ITU-R Recommendation BT.601 sampled video stream [1]), which will be denoted in this document as $Y(i, j, t)$, where $t = 1, 2, 3, ..., N$ (the total number of frames in the video clip) and i and j are the row and column indices of the images, respectively. The algorithm has two major components. The first component involves processing image pixels to produce a frame-by-frame motion energy time history for the video clip (Section 2.1). The second component involves examining this motion energy time history for dropped/repeated video frames (Section 2.2).

2.1. Computing the Motion Energy Time History

The motion energy time history of a video clip requires three processing steps on the sampled video images.

Step 1) Compute the Temporal Information (*TI*) difference sequence given by

$$TI(i,j,t) = Y(i,j,t) - Y(i,j,t-1),\ (i,j) \in SROI, and\ t = 2,3,...,N. \qquad (1)$$

Here *SROI* is the Spatial Region of Interest, which may be selected to be the central portion of the image to eliminate image border pixels that do not contain valid picture (e.g., some cameras may not fill the entire ITU-R Recommendation BT.601 frame, encoders may not transmit the entire frame).

Step 2) Zero image pixels in *TI* that have an amplitude less than or equal to an image motion threshold M_{image}.[2]

[2] There are a number of parameters (e.g., M_{image}) that control the behavior of the algorithm presented herein. Recommended values for these parameters will be given in Table 1.

$$TI(i,j,t) = \begin{cases} TI(i,j,t) & if\ abs(TI(i,j,t) > M_{image} \\ 0 & otherwise \end{cases} \quad (2)$$

This step eliminates low level noise from being counted as image motion. M_{image} may also be adjusted higher to eliminate motion pixels that fall below the ability to perceive them.

Step 3) Square *TI* to convert from amplitude to energy and compute the mean of each video frame. Here the resulting time history of frame-by-frame values that contain the motion energy will be represented as *TI2* and computed as:

$$TI2(t) = \underset{over\ i,j}{mean}\{TI(i,j,t)^2\}. \quad (3)$$

Figure 1 gives an example plot of *TI2* for a 40-frame segment of a 30 fps video clip that was derived from 24 fps film. Here, every 5th video frame is a frame repeat from the previous frame (evident by the *TI2* dips present at frames 5, 10, 15, and 20). At frame 23 there is a scene change (causing a large *TI2* spike) to a nearly still scene that continues for the remainder of the segment.

The next section will address how to examine the *TI2* waveform to locate dropped/repeated video frames.

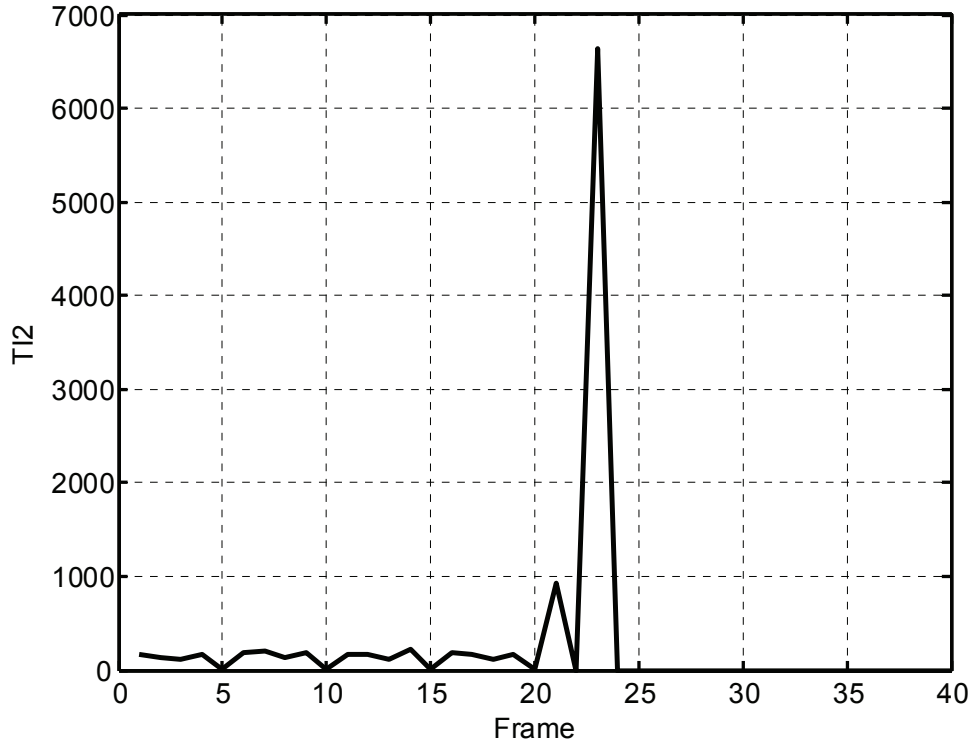

Figure 1. Example *TI2* time history.

2.2. Finding Dropped/Repeated Frames

For several reasons, locating dropped/repeated video frames in the *TI2* time history is more complicated than it might first appear. Scenes can vary from being still or nearly still (e.g., a zoomed out video of someone talking where only their lips are moving) to having very large amounts of motion (e.g., pan or zoom). Dropped frames can have small amounts of residual motion due to very minor changes in the image pixel values. Video compression systems can perform partial frame updates where only a small fraction of the video frame is updated. Thus, there is an element of subjective perception involved in determining the level of motion where these minor frame updates are counted as dropped video frames and not as new video frames.

Examination of video scenes from many different compression systems has produced a method which utilizes a dynamic threshold to determine dropped video frames. This threshold is raised for scenes with more motion and lowered for scenes with less motion. When more motion is present in the video scene, the dropped frames can contain more residual motion yet still be perceived as dropped frames. In addition, brief frame drops of one frame duration (called *dips* in this section) can contain even greater amounts of residual motion than a series of dropped frames.

In light of the above discussion, the algorithm will use a fixed motion energy threshold M_{drop} to determine frame *drops* and a different fixed motion energy threshold M_{dip} to determine frame *dips*, where the dips must have an amplitude of at least A_{dip} (i.e., the *TI2* waveform must be higher on both sides of the dip by at least A_{dip}). Then, a dynamic factor *dfact* will multiply the three fixed constants (M_{drop}, M_{dip}, and A_{dip}) and these dynamically adjusted thresholds will be used to determine the actual drops and dips in the *TI2* waveform. The dynamic factor *dfact* will be derived from the average level of motion energy that is present in the video clip.

We continue with the step by step numbering used in Section 2.1 to complete the algorithm description.

Step 4) Compute the average *TI2* value (*TI2_ave*) as

$$TI2\_ave = \underset{over\ k}{mean}\{TI2\_sort(k)\}, \quad ceil(F_{cut} * (N-1)) \le k \le floor((1-F_{cut}) * (N-1)) \quad (4)$$

Here, *TI2_sort* is the *TI2* vector from step 3 sorted from low to high, with new index *k* rather than *t* (recall that the *TI2* vector has *N*-1 samples). F_{cut} is the fraction of scene cuts to eliminate before computing the average (e.g., F_{cut} = 0.02 will eliminate 2 scene cuts every 100 frames), *ceil* and *floor* are rounding functions that round up and down to the nearest integer, respectively. Scene cuts cause very high *TI2* values and unduly influence *TI2_ave*, particularly for low motion scenes. Both low and high *TI2* points are eliminated so that the average *TI2* value is not influenced when scene cuts are absent.

Step 5) Compute the dynamic factor *dfact* as

$$\begin{aligned} &dfact = a + b * \log(TI2\_ave) \\ &if\ (dfact < c)\ then\ set\ dfact = c \end{aligned} \quad (5)$$

where *a*, *b*, and *c* are positive constants and log is the natural logarithm base *e*. The constant *c* is necessary to limit *dfact* to a small positive value. Equation 5 says that the perception of frame drops and dips is linearly dependent upon the log of the average motion energy. Frame drops for higher motion scenes can have a higher residual motion yet still be perceived as frame drops.

Step 6) Compute the Boolean variable *drops* (equal to 1 when a frame drop is detected, otherwise equal to 0) as

$$drops(t) = \begin{cases} 1 & if\ TI2(t) \le dfact * M_{drop} \\ 0 & otherwise \end{cases} \tag{6}$$

Step 7) Compute the Boolean variable *dips* (equal to 1 when a frame dip is detected, otherwise equal to 0) as

$$dips(t) = \begin{cases} 1 & if\ TI2(t) \le dfact * M_{dip}\ and\ dips\_mag(t) \ge dfact * A_{dip} \\ 0 & otherwise \end{cases} \tag{7}$$

where *dips_mag* is a function that finds the magnitude of the dips and is given by

$$\begin{aligned} &dips\_mag(t) = \min\{TI2(t-1) - TI2(t), TI2(t+1) - TI2(t)\} \\ &if\ (dips\_mag(t) < 0)\ then\ set\ dips\_mag(t) = 0 \end{aligned} \tag{8}$$

The endpoints (i.e., t =2 and t=N) of the *dips* vector in equation (7) are set equal to 0 since the *dips_mag* function is undefined for these two data points.

Step 8) Some frames may be classified as both a drop and a dip. Thus, the logical OR of the *drops* and *dips* vectors from steps 6 and 7 gives 1's for those frames in the video clip that are detected as dropped/repeated frames and 0's otherwise. One can compute the Fraction of Dropped Frames (*FDF*) by summing the elements in the vector and dividing by the maximum number of samples that could be detected as drops or dips, namely

$$FDF = \text{sum}(drops\ \text{OR}\ dips)/(N-3). \tag{9}$$

Step 9) (Optional for RR measurement.) If one has access to the corresponding time-aligned original source video clip, one can adjust the *FDF* of the destination video clip downward to account for still frames and/or detected drops and dips in the source clip (e.g., see the example shown in Figure 1). Here

$$\begin{aligned} &FDF_{RR} = \frac{FDF_{dest} - FDF_{source}}{1 - FDF_{source}} \\ &if\ (FDF_{source} > 0.9)\ then\ FDF_{RR}\ is\ undefined \\ &if\ (FDF_{RR} < 0)\ then\ set\ FDF_{RR} = 0 \end{aligned} \tag{10}$$

where FDF_{dest} and FDF_{source} are the *FDF*s of the destination and source video clips, respectively. The divisor (1 - FDF_{source}) is required since detected drops in the source video clip reduce the

total number of samples that are available for estimating FDF_{RR}. If too many drops are detected in the source video clip (i.e., $FDF_{source} > 0.9$), then the divisor (1 - FDF_{source}) will approach zero. In this case, it would be prudent to treat FDF_{RR} as not having a value (i.e., undefined). Finally, FDF_{RR} needs to be limited at zero because the algorithm may detect drops and dips in the source video clip but not in the destination video clip.

Recommended values of the parameters that control the behavior of the algorithm are given in Table 1. These recommended values were obtained by examining plots of the *TI2* waveform and its associated drops and dips for different types of data sets that contained a wide variety of video encoders and transmission errors. When there was some doubt as to whether or not certain video frames should be classified as drops or dips, the video clips themselves were examined using frame-by-frame playback. While this approach is subjective, it produces a rapid convergence to parameter values that work well across many video clips, and indeed was the method used to determine the need for dynamic thresholds (step 5). A more optimal approach would be to have a set of human observers classify the drops and dips present in each video clip and then utilize a mathematical search to optimize the algorithm's parameters. However, this approach is very costly in terms of human labor.[3]

Table 1. Recommended Values for Algorithm Parameters

Parameter	Value
M_{image}, Equation (2)	30
F_{cut}, Equation (4)	0.02
a, Equation (5)	2.5
b, Equation (5)	1.25
c, Equation (5)	0.1
M_{drop}, Equation (6)	0.015
M_{dip}, Equation (7)	1.0
A_{dip}, Equation (7)	3.0

[3] An estimate of the algorithm's performance is not provided in this document since the algorithm has not yet been exhaustively tested. Therefore, the recommended values given in Table 1 should be considered preliminary.

Figure 2 gives the *TI2* plot of Figure 1 with detected drops (shown in red circles) and detected dips (shown in green squares) using the recommended parameter values given in Table 1. The detected drops result from the scene being a still scene from frames 24 to 40. The detected dips at frames 5, 10, 15, and 20 result from the film conversion process described earlier where every 5th frame was repeated. Visual examination of frame 22 (immediately prior to the scene cut at frame 23) reveals that this is indeed a frame repeat of prior video frame 21. Thus, frame 22 is correctly detected as a drop by the algorithm.

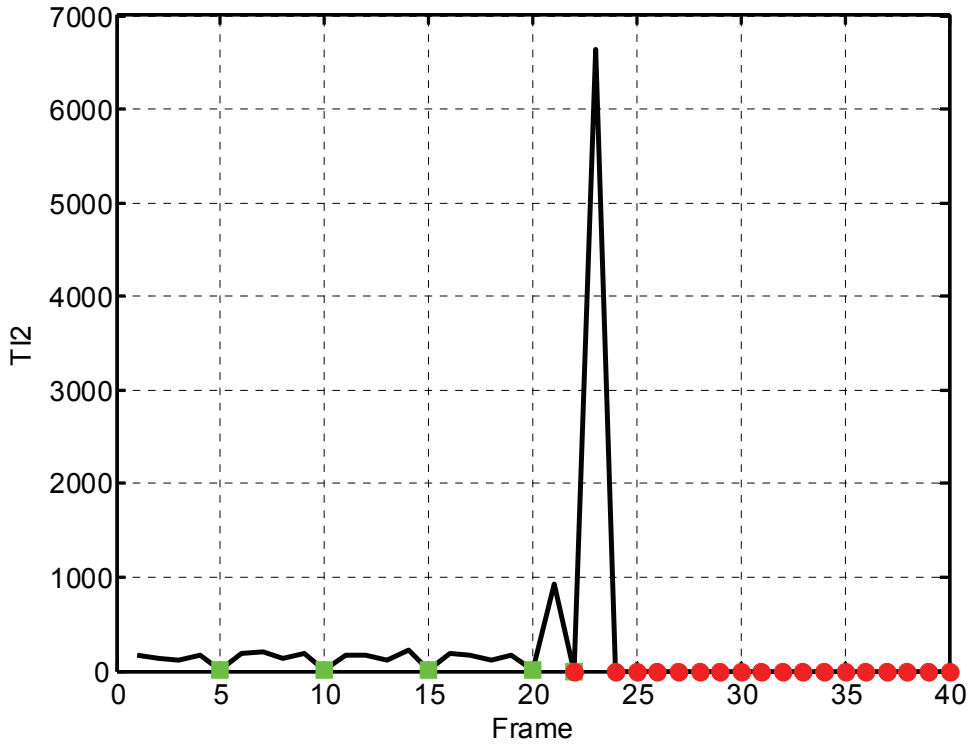

Figure 2. The *TI2* plot of Figure 1 with detected drops (red) and dips (green) shown.

3. EXAMPLE RESULTS

This section shows example results for the Phase I Multi-Media (MM) VGA data sets from the Video Quality Experts Group (VQEG) [2]. These data sets were chosen since they contain a wide range of modern video compression algorithms (e.g., H.264) and network impairments (e.g., dropped IP packets). Altogether, there were 13 data sets of 166 clips each.

Figure 3 shows an example *TI2* waveform for a segment of a destination video clip with frame freezes (frame drops shown in red) resulting from dropped IP packets. This particular scene had very high motion (pan of a crowd at a football game). Error blocks were present in the video immediately before the frame freeze periods, and these error blocks show up as large spikes in the *TI2* waveform (at frames 6 and 10). The missing IP packets at frame 6 resulted in three dropped frames (i.e., frames 7, 8, and 9) while the missing IP packets at frame 10 resulted in only one dropped frame (i.e., frame 11). At frame 36, error blocks due to missing IP packets were also present in the video but a frame freeze period did not result from the dropped packets there.

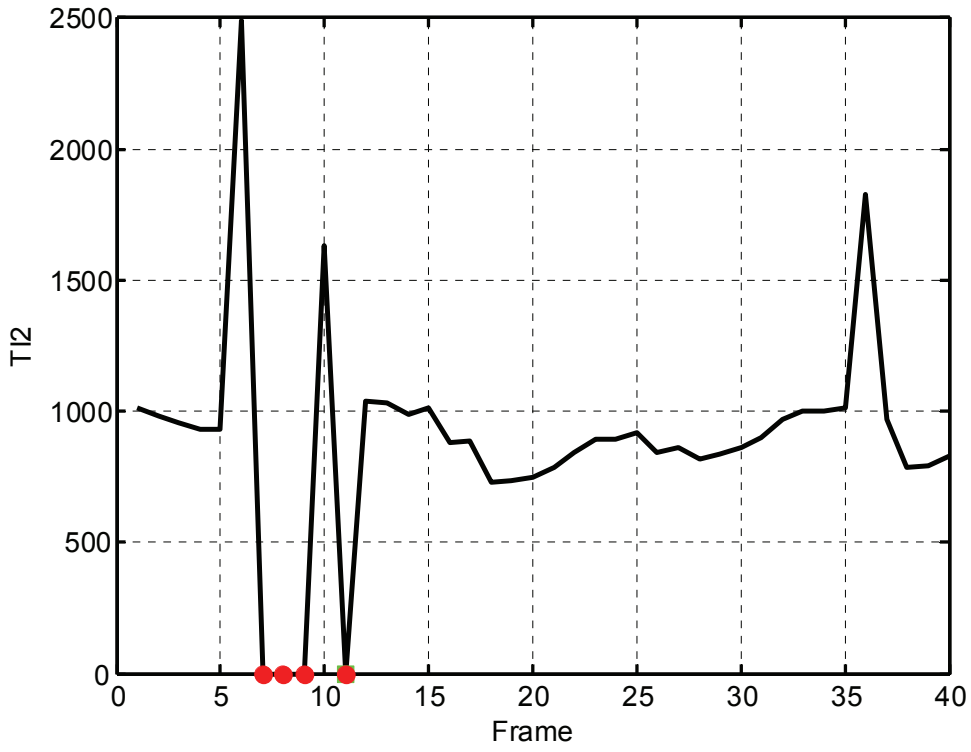

Figure 3. Example *TI2* waveform with error blocks and frame freezes due to dropped IP packets.

Figure 4 shows an example of a *TI2* waveform where the video coder performs dynamic frame dropping that depends upon the amount of motion in the video scene. Low motion portions of the scene are transmitted at full or near full frame rates (from frames 1 to 24, only one frame

drop occurred at frame 10) while high motion portions are transmitted at reduced frame rates by dropping selected video frames (from frames 25 to 40, frame drops occurred at frames 25, 28, 30, 32, 34, 36, 38, and 40). This is a common method used to save on transmission bits.

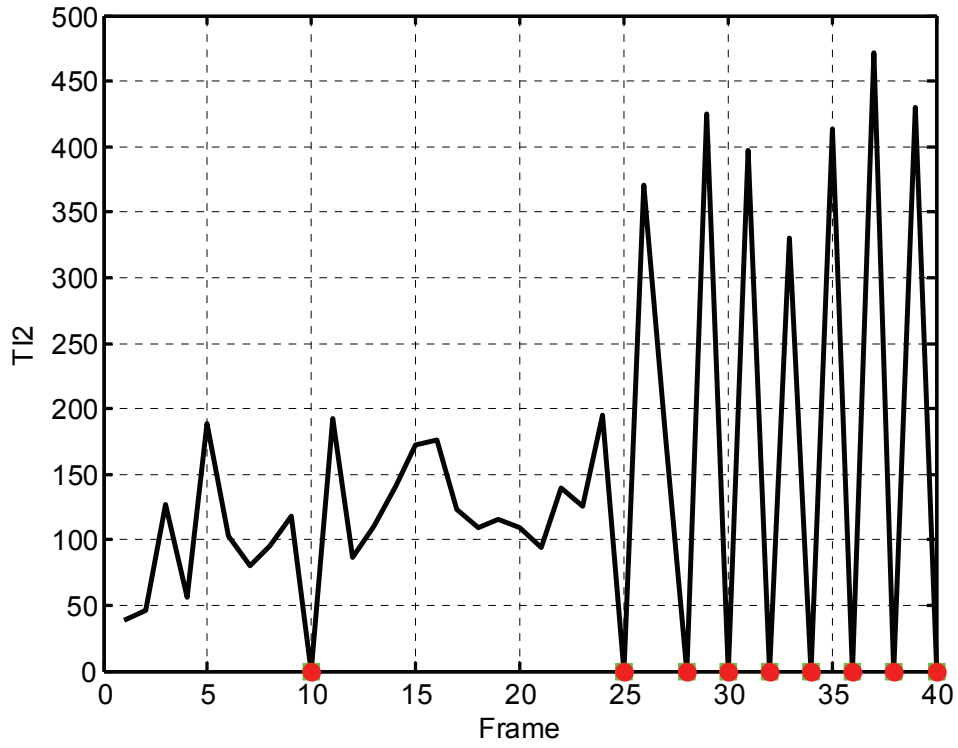

Figure 4. Example of *TI2* waveform with dynamic frame dropping.

Figure 5 shows an interesting example of a video coder that performs partial frame updates every other video frame. Here, the M_{dip} and A_{dip} thresholds that control the detection of the dips (shown in green) determine when the partial frame updates are too small to be true frame updates, and are hence declared to be dips. For this video clip, frame dips are detected before frame 36, which is detected as a frame drop (shown in red). Visual frame-by-frame examination of the video clip confirms this behavior where the detected dips (shown in green) are very minor frame updates (too minor to be viewed as new video frames) and the three detected drops (shown in red at frames 36, 38, and 43) are true frame repeats which are identical to the previous frames.

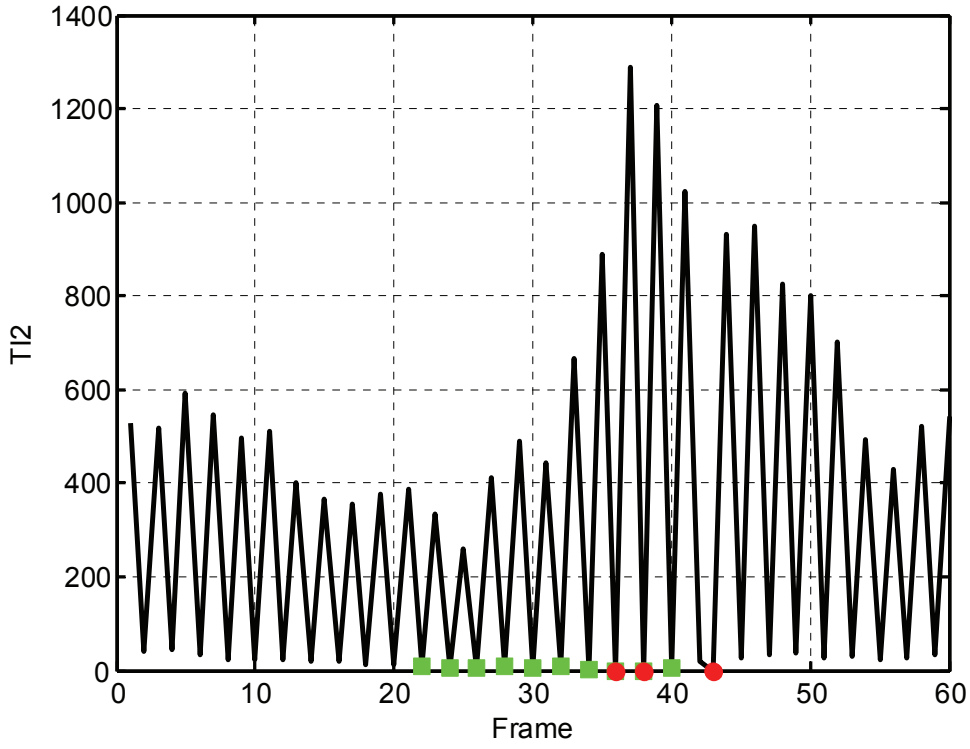

Figure 5. Example of *TI2* waveform with partial frame updates.

No thresholding scheme is perfect and the method presented in this document also has its problems. Figure 6 shows a *TI2* waveform for an original source scene of a head and shoulders shot where only the lips and eyelids are moving (note the extremely low magnitude of *TI2*). The scene has periods of very low motion which are mistakenly detected as frame drops (frames 29 to 31). Comparison of Figure 7 (from a video system that drops frames) with Figure 6 demonstrates one advantage of using the RR version of *FDF* given by equation (10), namely, compensation for mistakes made by the algorithm for low motion scenes.

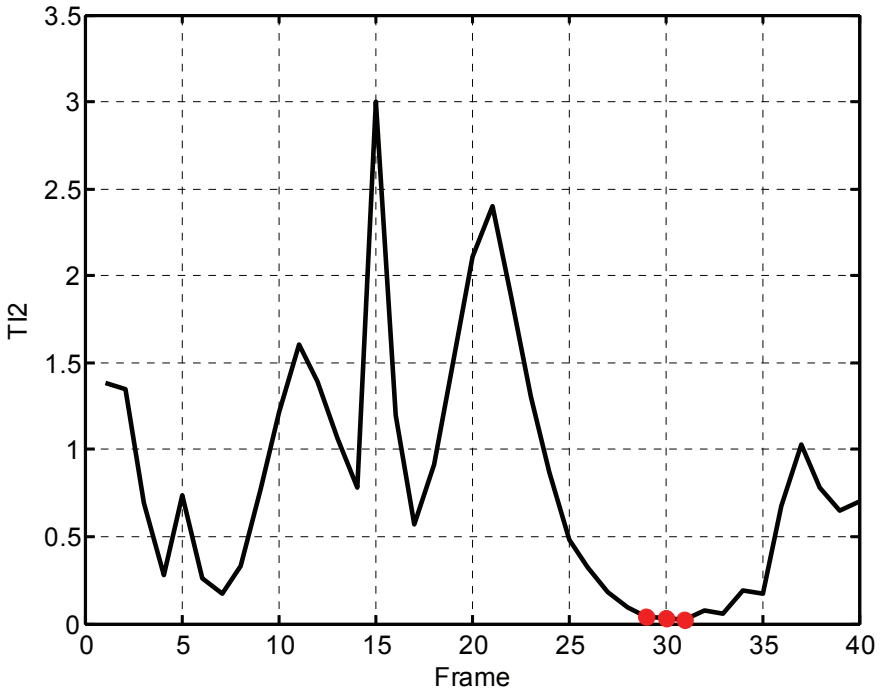

Figure 6. Example of *TI2* waveform for original scene with low motion.

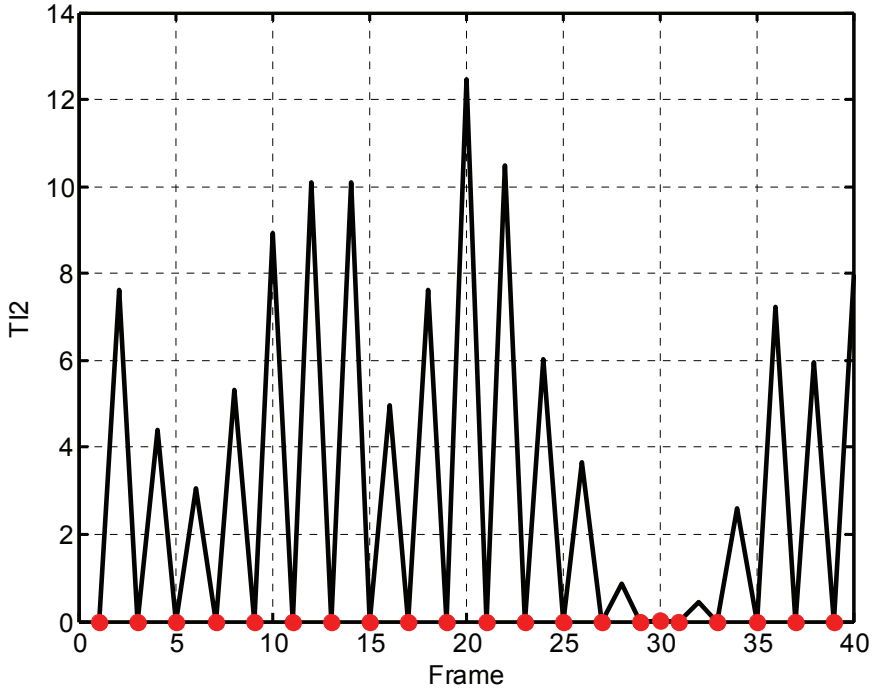

Figure 7. Example of corresponding destination video *TI2* waveform for Figure 6.

4. MATLAB CODE

This section provides MATLAB® code to compute the NR and RR versions of *FDF*. This code assumes that the user is processing a set of video clips that are all present in a given directory. The video clips must be named by test, scene, and HRC (an acronym for Hypothetical Reference Circuit, which is a video system that includes a video encoder, a digital transmission system, and a video decoder). The video clips must have the naming convention "test_scene_hrc.yuv" if stored as raw big-YUV files[4] or "test_scene_hrc.avi" if stored as uncompressed UYVY AVI files. For the original clip names "hrc" must be "original" (required for RR version of *FDF*).

The main routine is called "est_hrc_fdf_rr " and is given in Section 4.1. This routine will calculate the *FDF* of each video clip one HRC at a time, starting with the original HRC. The average *FDF* of each HRC is also calculated and displayed. The function returns all the results in a MATLAB struct array (see code for a complete description of the returned results).

Also included in Sections 4.2 and 4.3 is code to read big-YUV files and to find dips in the *TI2* waveform. These functions are called "read_bigyuv" and "findpeaks2" respectively (finding dips is equivalent to finding peaks in the negative waveform).

4.1. Function to Calculate *FDF*

```
function [results] = est_hrc_fdf_rr(clip_dir, test, varargin);
% EST_HRC_FDF_RR
%   Estimate the fraction dropped frames (fdf) of all HRCs in a video test
%   where the video clips are stored in the specified directory.  The video
%   clips must have names that conform to the standard naming convention
%   (test_scene_hrc.avi or test_scene_hrc.yuv, with no extra '_' or '.' in
%   the file names).  If AVI files are used, they must be in the 'YCbCr'
%   format.
% SYNTAX
%   [results] = est_hrc_fdf_rr('clip_dir', 'test', option);
% DESCRIPTION
%   This function will process all video clips in the user specified
%   clip_dir and test, estimate the average fdf of each clip, and then
%   average these clip results to produce an estimate for each HRC.  The
%   algorithm is a zero reference algorithm that computes the Temporal
%   Information (TI) difference between successive frames, zeros low motion
%   pixels, sums the TI^2 energy of the highest motion pixels remaining in
%   each frame, and then examines these time history waveforms to locate
%   dropped frames.  The algorithm can be run in RR mode provided the
%   original clips are also present in the specified directory and their
%   names have the form test_scene_original.yuv or test_scene_original.avi.
%
%   The [results] returned by the function is a struct that contains the
%   following fields:
%       'test'      The test name for the video clip.
%       'scene'     The scene name for the video clip.
%       'hrc'       The hrc name for the video clip.
%       'ti2'       The TI^2 waveform for the video clip.
%       'drops'     The indices of the drops and dips in the TI^2 waveform.
%       'fdf'       The fraction of dropped frames for the video clip.
%
%   Any or all of the following optional properties may be requested (the
%   first option is required for yuv files, but not for avi files since
```

[4] A raw big-YUV file stores 4:2:2 YC_BC_R sampled video pixels (of one byte each) stored one row after another, where the first row of pixels are ordered as C_{B1}, Y_1, C_{R1}, Y_2, C_{B3}, Y_3, C_{R3}, Y_4, etc.

```
%   this information is read from the avi header).
%
%   'yuv',rows,cols,fps    Specifies the number of rows, cols, and frames
%                          per second (fps) of the yuv files.
%
%   'sroi',top,left,bottom,right,   Only use the specified spatial region
%                                   of interest (sroi) for the temporal
%                                   difference calculation.  By default,
%                                   all of the image is used.  The sroi is
%                                   inclusive, where top/left start at 1.
%
%   'frames',fstart,fstop  Only use the frames from fstart to fstop
%                          (inclusive) to perform the fdf estimate.
%                          By default, all frames will be used - this may
%                          cause "out of memory" errors for resolutions
%                          larger than QVGA or CIF on some computers.
%
%   'rr'    Operate in reduced reference mode where the number of dropped
%           frames in the processed clip is reduced by the number of
%           dropped frames in the original clip for the fdf calculation.
%
%   'verbose'   Display output and plots during processing.
%
% EXAMPLES
%   results = est_hrc_fdf_rr('clip_dir','test');
%   results = est_hrc_fdf_rr('clip_dir','test','yuv',486,720,30,...
%                'sroi',21,21,466,700,'verbose');
%

% Parameters that control the dropped frames detection algorithm.

% TI image pixels at or below this magnitude are not considered motion and
% are set equal to zero.  Thus, a pixel must have a TI difference magnitude
% greater than this level to contribute to the motion energy of the frame.
m_image = 30;

% If the average TI^2 value of a frame is less than or equal to a motion
% threshold, m_drop*dfact (a dynamic factor that is a function of the mean
% TI^2 value computed over all frames in the video sequence), then that
% frame is declared a frame repeat.
m_drop = 0.015;

% Simple thresholding does not catch all the perceptual dropped frames when
% there is some residual motion (small blocks being updated) since the TI^2
% waveform does not dip below the motion threshold.  If this condition
% occurs for single frame drops, they can be detected with a dip
% detector.  In order to be declared a valid dip, these negative 1-frame
% width dips must have a dip amplitude of at least a_dip*dfact, and the
% average TI^2 value for the frame must be less than or equal to
% m_dip*dfact.
a_dip = 3;
m_dip = 1;

% Constants that control the computation of the dynamic factor (dfact)
a = 2.5;  % DC shift
b = 1.25;  % Gain
c = 0.1;  % low end limit

% Fraction of scene cuts to eliminate before computing TI2_ave
f_cut = 0.02;

% Disable output warnings
warning off all;

% Add extra \ in clip_dir in case user forgot
clip_dir = strcat(clip_dir,'\');

% Validate input arguments and set their defaults
is_yuv = 0;
is_whole_image = 1;
is_whole_time = 1;
```

```
is_rr = 0;
verbose = 0;
cnt=1;
while cnt <= length(varargin),
    if ~ischar(varargin{cnt}),
        error('Property value passed into est_hrc_fdf_rr is not recognized');
    end
    if strcmpi(varargin(cnt),'yuv') == 1
        rows = varargin{cnt+1};
        cols = varargin{cnt+2};
        fps = varargin{cnt+3};
        is_yuv = 1;
        cnt = cnt + 4;
    elseif strcmpi(varargin(cnt),'sroi') == 1
        top = varargin{cnt+1};
        left = varargin{cnt+2};
        bottom = varargin{cnt+3};
        right = varargin{cnt+4};
        is_whole_image = 0;
        cnt = cnt + 5;
    elseif strcmpi(varargin(cnt),'frames') == 1
        fstart = varargin{cnt+1};
        fstop = varargin{cnt+2};
        nframes = fstop-fstart+1;
        is_whole_time = 0;
        cnt = cnt + 3;
    elseif strcmpi(varargin(cnt),'rr') == 1
        is_rr = 1;
        cnt = cnt + 1;
    elseif strcmpi(varargin(cnt),'verbose') == 1
        verbose = 1;
        cnt = cnt + 1;
    else
        error('Property value passed into est_hrc_fdr_rr not recognized');
    end
end

files = dir(clip_dir);  % first two files are '.' and '..'
num_files = size(files,1);

% Find the HRCs and their associated scenes
hrc_list = {};
scene_list = {};
for i=3:num_files
    this_file = files(i).name;
    und = strfind(this_file,'_'); % find underscores and period
    dot = strfind(this_file,'.');
    if(und) % possible standard naming convention file found
        this_test = this_file(1:und(1)-1);  % pick off the test name
        if(strmatch(test,this_test,'exact'))  % test clip found
            this_scene = this_file(und(1)+1:und(2)-1);
            this_hrc = this_file(und(2)+1:dot(1)-1);
            % See if this HRC already exists and find its list location
            loc = strmatch(this_hrc,hrc_list,'exact');
            if(loc)  % HRC already present, add to scene list for that HRC
                scene_list{loc} = [scene_list{loc} this_scene];
            else  % new HRC found
                hrc_list = [hrc_list;{this_hrc}];
                this_loc = size(hrc_list,1);
                scene_list(this_loc) = {{this_scene}};
            end
        end
    end
end

scene_list = scene_list';
num_hrcs = size(hrc_list,1);

%  Reorganize the HRC and scene lists to process the originals first
orig_loc = strmatch('original',hrc_list,'exact');
if (orig_loc > 1)
```

```
    top_hrc_list = [hrc_list(orig_loc);hrc_list(1:orig_loc-1)];
    top_scene_list = [scene_list(orig_loc);scene_list(1:orig_loc-1)];
else
    top_hrc_list = hrc_list(orig_loc);
    top_scene_list = scene_list(orig_loc);
end
if (orig_loc < num_hrcs)
    hrc_list = [top_hrc_list; hrc_list(orig_loc+1:num_hrcs)];
    scene_list = [top_scene_list; scene_list(orig_loc+1:num_hrcs)];
else
    hrc_list = top_hrc_list;
    scene_list = top_scene_list;
end

%Results struct
results = struct('test', {}, 'scene', {}, 'hrc', {}, 'ti2', {}, ...
    'drops', {}, 'fdf', {});

% Process one HRC at a time to compute average fdf
index = 1;  % index used to store results

% Hold the number of drops and dips in the original for RR mode of operation
orig_scene_list = {};  % contains the original scene name
orig_dropnum = [];  % contains the number of dropped frames in the original
orig_index = 0;  % index used to access original dropped frame information

for i = 1:num_hrcs

    fdf_ave = 0;  % initialize the fdf average summer for this HRC
    fdf_scenes = 0;  % initialize number of scenes summer for this HRC
    this_hrc = hrc_list{i};
    num_scenes = size(scene_list{i},2);  % number of scenes in this HRC

    for j = 1:num_scenes
        this_scene = scene_list{i}{j};
        results(index).test = test;
        results(index).scene = this_scene;
        results(index).hrc = this_hrc;

        % Read video file
        if (~is_yuv)  % AVI file

            % Re-generate the processed avi file name
            proc = strcat(clip_dir, test,'_', this_scene, '_', this_hrc, '.avi');
            [avi_info] = read_avi('Info',proc);
            rows = avi_info.Height;
            cols = avi_info.Width;

            % Set/Validate the ROI
            if (is_whole_image)
                top = 1;
                left = 1;
                bottom = rows;
                right = cols;
            elseif (top<1 || left<1 || bottom>rows || right>cols)
                display('Requested ROI too large for image size.\n');
                return;
            end

            fps = avi_info.FramesPerSecond;  % frames per second of input file
            tframes = avi_info.NumFrames;  % total frames in file

            % Set/Validate the time segment to use
            if (is_whole_time)
                fstart= 1;
                fstop = tframes;
                nframes = tframes;
            elseif (fstart<1 || fstop>tframes)
                display('Requested temporal segment is too large for file size.\');
                return;
            end
```

```
    % Read in video and clear color planes to free up memory
    [y,cb,cr] = read_avi('YCbCr',proc,'frames',fstart,fstop,...
        'sroi',top,left,bottom,right);
    clear cb cr;

else  % YUV file

    % Re-generate the processed YUV file name
    proc = strcat(clip_dir, test,'_', this_scene, '_', this_hrc, '.yuv');

    % Set/Validate the ROI
    if (is_whole_image)
        top = 1;
        left = 1;
        bottom = rows;
        right = cols;
    elseif (top<1 || left<1 || bottom>rows || right>cols)
        display('Requested ROI too large for image size.\n');
        return;
    end

    % Find the total frames of the input file
    [fid, message] = fopen(proc, 'r');
    if fid == -1
        fprintf(message);
        error('Cannot open this clip''s bigyuv file, %s', proc);
    end
    % Find last frame.
    fseek(fid,0, 'eof');
    tframes = ftell(fid) / (2 * rows * cols);
    fclose(fid);

    % Set/Validate the time segment to use
    if (is_whole_time)
        fstart= 1;
        fstop = tframes;
        nframes = tframes;
    elseif (fstart<1 || fstop>tframes)
        display('Requested temporal segment is too large for file size.\n');
    end

    % Read in video and clear color planes to free up memory
    [y,cb,cr] = read_bigyuv(proc,'frames',fstart,fstop,...
        'size',rows,cols,'sroi',top,left,bottom,right);
    clear cb cr;
end

% Compute TI over the sroi and time segment
ti = y(:,:,1+1:nframes)-y(:,:,1:nframes-1);
clear y;
[nrows, ncols, nsamps] = size(ti);

%  Zero pixels below m_image (image motion threshold)
ti((abs(ti) <= m_image)) = 0.0;

%  Convert to energy and take mean of each frame
ti2 = mean(reshape(ti,nrows*ncols,nsamps).^2);
clear ti;
results(index).ti2 = ti2;

% Calculate the average TI^2 value for the clip, but eliminate
% scene cuts and
ti2_sort = sort(ti2);
ti2_ave = mean(ti2_sort(ceil(f_cut*nsamps):floor((1-f_cut)*nsamps)));

% dynamic thresholding scheme based on ti2_ave
dfact = a+b*log(ti2_ave);

% Limit dfact to positive numbers
if (dfact < c)
```

```
    dfact = c;
end

% Compute the dynamic thresholds for this video clip
this_m_drop = dfact*m_drop;
this_a_dip = dfact*a_dip;
this_m_dip = dfact*m_dip;

%  Find dropped frames that fall below motion threshold
drops = find(ti2 <= this_m_drop);

%  Find additional drops with dip detector, which is the same as finding
%  the peaks of the negative waveform.
[dips_mag, dips] = findpeaks2(-1*ti2,'threshold',this_a_dip);
dips_mag = -1*dips_mag;

% Eliminate dips that are above this_m_dip
dips = dips(find(dips_mag <= this_m_dip));

% Calculate total of drops and dips, eliminating redundancies
drops_dips = union(drops,dips);
results(index).drops = drops_dips;

% Check if original and save number of dropped frames
if (strmatch('original',this_hrc,'exact'))
    orig_index = orig_index+1;  % index for this original clip
    orig_scene_list{orig_index} = this_scene;
    orig_dropnum(orig_index) = length(drops_dips);
end

% calculate the fraction of dropped frames
this_fdf = length(drops_dips)/(nsamps-2);
this_fps = fps*(1-this_fdf);  % convert to fps if desired
if (is_rr)
    % Find the corresponding original dropnum and compensate fdf
    this_match = strmatch(this_scene,orig_scene_list,'exact');
    orig_fdf = orig_dropnum(this_match)/(nsamps-2);
    if (orig_fdf > 0.9)
        this_fdf = 'NaN';
        this_fps = 'NaN';
    else
        this_fdf = max(0,(this_fdf-orig_fdf)/(1-orig_fdf));
        this_fps = fps*(1-this_fdf);
    end
end

% Store the fdf information for this clip and update HRC average
results(index).fdf = this_fdf;
index = index+1;
if (~ischar(this_fdf))
    fdf_ave = fdf_ave + this_fdf;
    fdf_scenes = fdf_scenes+1;
end

pause(0.1);  % to allow user to stop program

if(verbose)
    plot(ti2, 'k', 'LineWidth', 2);
    grid on;
    set(gca,'LineWidth',1)
    set(gca,'FontName','Ariel')
    set(gca,'fontsize',12)
    hold on
    plot(dips, ti2(dips), '.g', 'markersize', 10);
    plot(drops, ti2(drops), '.r', 'markersize', 10);
    xlabel('Frame');
    ylabel('TI2');
    title('Detected Frame Drops and Dips in Red and Green');
    hold off
    if (~ischar(this_fdf))
        fprintf('Test = %s,   Scene = %s,   HRC = %s,   fdf = %4.2f\n',...
```

```
                    test, this_scene, this_hrc, this_fdf);
            else
                fprintf('Test = %s,   Scene = %s,   HRC = %s,   fdf = %s\n',...
                    test, this_scene, this_hrc, this_fdf);
            end
            pause(0.1)
        end
    end

    %  Compute average fdf for this HRC if it is available
    if (fdf_scenes >= 1)
        fdf_ave = fdf_ave/fdf_scenes;
        if (verbose)
            fprintf('HRC = %s, fdf_ave = %4.2f\n', this_hrc, fdf_ave);
        end
    end
end
```

4.2. Function to Read Big-YUV files

```
function [y,cb,cr] = read_bigyuv(file_name, varargin);
% READ_BIGYUV
%   Read images from bigyuv-file.
% SYNTAX
%   [y] = read_bigyuv(file_name);
%   [y,cb,cr] = read_bigyuv(...);
%   [...] = read_bigyuv(...,'PropertyName',PropertyValue,...);
% DESCRIPTION
%   Read in images from bigyuv file named 'file_name'.
%
%   The luminance plane is returned in 'Y'; the color planes are
%   returned in 'cb' and 'cr' upon request.  The Cb and Cr color planes
%   will be upsampled by 2 horizontally.
%
%   The following optional properties may be requested:
%
%   'sroi',top,left,bottom,right,
%                        Spatial region of interest to be returned.  By default,
%                        the entirety of each image is returned.
%                        Inclusive coordinates (top,left),(bottom,right) start
%                        numbering with row/line number 1.
%   'size',row,col,    Size of images (row,col).  By default, row=486,
%                           col=720.
%   'frames',start,stop,    Specify the first and last frames, inclusive,
%                                  to be read ('start' and 'stop').  By
%                                  default, the first frame is read.
%   '128'        Subtract 128 from all Cb and Cr values.  By default, Cb
%                       and Cr values are left in the [0..255] range.
%   'interp'           Interpolate Cb and Cr values.  By default, color
%                        planes are pixel replicated.  Note:  Interpolation is slow.
%
%   Color image pixels will be pixel replicated, so that Cb and Cr images
%   are not subsampled by 2 horizontally.

% read values from clip_struct that can be over written by variable argument
% list.
is_whole_image = 1;
is_sub128 = 0;
is_interp = 0;

num_rows = 486;
num_cols = 720;

start = 1;
stop = 1;

% parse varargin list (property values)
```

```
cnt = 1;
while cnt <= nargin - 1,
    if ~isstr(varargin{cnt}),
        error('Property value passed into bigyuv_read not recognized');
    end
    if strcmp(lower(varargin(cnt)),'sroi') == 1,
        sroi.top = varargin{cnt+1};
        sroi.left = varargin{cnt+2};
        sroi.bottom = varargin{cnt+3};
        sroi.right = varargin{cnt+4};
        is_whole_image = 0;
        cnt = cnt + 5;
    elseif strcmp(lower(varargin(cnt)),'size') == 1,
        num_rows = varargin{cnt+1};
        num_cols = varargin{cnt+2};
        cnt = cnt + 3;
    elseif strcmp(lower(varargin(cnt)),'frames') == 1,
        start = varargin{cnt+1};
        stop = varargin{cnt+2};
        cnt = cnt + 3;
    elseif strcmp(lower(varargin(cnt)),'128') == 1,
        is_sub128 = 1;
        cnt = cnt + 1;
    elseif strcmp(lower(varargin(cnt)),'interp') == 1,
        is_interp = 1;
        cnt = cnt + 1;
    else
        error('Property value passed into bigyuv_read not recognized');
    end
end

if mod(num_cols,2) ~= 0,
    fprintf('Error: number of columns must be an even number.\nThis 4:2:2 format stores 4 bytes
for each 2 pixels\n');
    error('Invalid specification for argument "num_cols" in read_bigyuv');
end

% Open image file
% [test_struct.path{1} clip_struct.file_name{1}]
[fid, message] = fopen(file_name, 'r');
if fid == -1
    fprintf(message);
    error('bigyuv_read cannot open this clip''s bigyuv file, %s', file_name);
end

% Find last frame.
fseek(fid,0, 'eof');
total = ftell(fid) / (2 * num_rows * num_cols);
if stop > total,
    error('Requested a frame past the end of the file.  Only %d frames available', total);
end
if stop < 0,
    error('Range of frames invalid');
end
if start > stop | stop < 1,
    error('Range of frames invalid, or no images exist in this bigyuv file');
end

% find range of frames requested.
prev_tslice_frames = start - 1;
tslice_frames = stop - start + 1;
number = start;

% go to requested location
if isnan(start),
    error('first frame of this clip is undefined (NaN).');
end
offset = prev_tslice_frames * num_rows * num_cols * 2; %pixels each image
status = fseek(fid, offset, 'bof');

if status == -1,
```

```
    fclose(fid);
    error('bigyuv_read cannot seek requested image location');
end

% initialize memory to hold return images.
y = zeros(num_rows,num_cols,tslice_frames, 'single');

if (nargout == 3),
    cb = y;
    cr = y;
end

% loop through & read in the time-slice of images
this_try = 1;
for cnt = 1:tslice_frames,
    where = ftell(fid);
    [hold_fread,count] = fread(fid, [2*num_cols,num_rows], 'uint8=>uint8');
    if count ~= 2*num_cols*num_rows,
        % try one more time.
        fprintf('Warning: bigyuv_read could not read entirety of requested image time-slice; re-
trying\n');
        %pause(5);
        if where == -1,
            fprintf('Could not determine current location.  Re-try failed.\n');
            error('bigyuv_read could not read entirety of requested image time-slice');
            fclose(fid);
        end
        fseek(fid, where, 'bof');
        [hold_fread,count] = fread(fid, [2*num_cols,num_rows], 'uint8=>uint8');
        if count ~= 2*num_cols*num_rows,
            fclose(fid);
            hold = sprintf('time-slice read failed for time-slice in %s\nbigyuv_read could not
read entirety of requested time-slice', file_name);
            error(hold);
        end
    end

    % pick off the Y plane (luminance)
    temp = reshape(hold_fread', num_rows, 2, num_cols);
    uncalib = squeeze(temp(:,2,:));
    y(:,:,cnt) = single(uncalib);

    % If color image planes are requested, pick those off and perform
    % pixel replication to upsample horizontally by 2.
    if nargout == 3,
        temp = reshape(hold_fread,4,num_rows*num_cols/2);

        color = reshape(temp(1,:),num_cols/2,num_rows)';
        color2 = [color ; color];
        uncalib = reshape(color2,num_rows,num_cols);
        cb(:,:,cnt) = single(uncalib);
        if is_sub128,
            cb(:,:,cnt) = cb(:,:,cnt) - 128;
        end

        color = reshape(temp(3,:),num_cols/2,num_rows)';
        color2 = [color ; color];
        uncalib = reshape(color2,num_rows,num_cols);
        cr(:,:,cnt) = single(uncalib);
        if is_sub128,
            cr(:,:,cnt) = cr(:,:,cnt) - 128;
        end

        % Interpolate, if requested
        if is_interp == 1,
            for i=2:2:num_cols-2,
                cb(:,i,cnt) = (cb(:,i-1,cnt) + cb(:,i+1,cnt))/2;
                cr(:,i,cnt) = (cr(:,i-1,cnt) + cr(:,i+1,cnt))/2;
            end
        end
    end
```

```
end

fclose(fid);

if ~is_whole_image,
    y = y(sroi.top:sroi.bottom, sroi.left:sroi.right, :);
    if nargout == 3,
        cb = cb(sroi.top:sroi.bottom, sroi.left:sroi.right, :);
        cr = cr(sroi.top:sroi.bottom, sroi.left:sroi.right, :);
    end
end
```

4.3. Function to Find Dips in *TI2* Waveform

```
function [mags, locs] = findpeaks2(x, varargin);
% FINDPEAKS2
%   Find the peaks in vector x.
% SYNTAX
%   [mags, locs] = findpeaks2(x, 'threshold', thres);
% DESCRIPTION
%   This function will find all peaks in vector x and return their
%   magnitudes (mags) and locations (locs).  Note that the first and last
%   element in the vector will never be declared a peak.
%
%   Any or all of the following optional properties may be requested.
%
%   'threshold',thres    Specifies the threshold for the peak finder.  The
%                        current peak must be at least this much above the
%                        neighboring values to be declared a peak.  The
%                        default value for thres is zero.
%

% Validate input arguments and set their defaults
thres = 0;
cnt = 1;
while cnt <= length(varargin),
    if ~isstr(varargin{cnt}),
        error('Property value passed into findpeaks2 is not recognized');
    end
    if strcmpi(varargin(cnt),'threshold') == 1
        thres = varargin{cnt+1};
        cnt = cnt + 2;
    else
        error('Property value passed into findpeaks2 not recognized');
    end
end
n = length(x);
d = diff(x);
locs = 1+find(d(2:n-1)<-1.0*thres & d(1:n-2)>thres);
if (length(locs) ~= 0)
    mags = x(locs);
else
    locs = [];
    mags = [];
end
```

5. CONCLUSION

Reliable NR metrics for quantifying perceptual aspects of video quality are difficult to develop. This document has presented an algorithm for quantifying one perceptual dimension of quality, namely, interruptions to the flow of motion in the video scene. This metric measures the fraction of dropped frames in a video clip by examining its frame-by-frame motion energy time history, a computationally efficient process that should lend itself well to real time quality monitoring systems.

The problem of correctly identifying repeated/dropped frames is more difficult than it might first appear. This is because a repeated frame need not be an exact binary replicate of the prior frame (e.g., video compression systems can perform partial frame updates). This document presents a dynamic thresholding scheme to identify these dropped frames. The algorithm works well for a wide range of video coders, transmission channels, and decoders.

One potential problem with the NR algorithm is that periods of low motion (e.g., moving lips) may be detected as dropped frames. A RR version of the algorithm can overcome this problem since the fraction of dropped frames of the destination video clip can be adjusted downward to account for still frames and/or detected drops in the source video (e.g., that might result from 24 fps film to 30 fps video conversion).

6. REFERENCES

[1] ITU-R Recommendation BT.601, “Studio encoding parameters of digital television for standard 4:3 and wide screen 16:9 aspect ratios," Recommendations of the ITU, Radiocommunication Sector.

[2] "Final Report from the Video Quality Experts Group on the Validation of Objective Models of Multimedia Quality Assessment, Phase I," available at http://www.its.bldrdoc.gov/vqeg/projects/multimedia/.

FORM **NTIA-29** U.S. DEPARTMENT OF COMMERCE
(4-80) NAT'L. TELECOMMUNICATIONS AND INFORMATION ADMINISTRATION

BIBLIOGRAPHIC DATA SHEET

1. PUBLICATION NO. TM-09-456	2. Government Accession No.	3. Recipient's Accession No.
4. TITLE AND SUBTITLE A No Reference (NR) and Reduced Reference (RR) Metric for Detecting Dropped Video Frames		5. Publication Date October 2008 6. Performing Organization NTIA/ITS.T
7. AUTHOR(S) Stephen Wolf		9. Project/Task/Work Unit No. 3141012
8. PERFORMING ORGANIZATION NAME AND ADDRESS Institute for Telecommunication Sciences National Telecommunications & Information Administration U.S. Department of Commerce 325 Broadway Boulder, CO 80305		10. Contract/Grant No.
11. Sponsoring Organization Name and Address National Telecommunications & Information Administration Herbert C. Hoover Building 14th & Constitution Ave., NW Washington, DC 20230		12. Type of Report and Period Covered
14. SUPPLEMENTARY NOTES		

15. ABSTRACT

Digital video transmission systems consisting of a video encoder, a digital transmission method (e.g., Internet Protocol – IP), and a video decoder can produce pauses in the video presentation that result from dropped or repeated video frames. For example, a common response of a video decoder to dropped IP packets is to momentarily freeze the video by repeating the last good video frame. This document presents a No Reference (NR) metric and a Reduced Reference (RR) metric for detecting these dropped video frames. These metrics may have application for in-service video quality monitoring.

16. Key Words

dropped; frames; metrics; No Reference (NR); Reduced Reference (RR); video

17. AVAILABILITY STATEMENT	18. Security Class. (This report)	20. Number of pages
☐ UNLIMITED.	Unclassified	
	19. Security Class. (This page)	21. Price:
	Unclassified	

NTIA FORMAL PUBLICATION SERIES

NTIA MONOGRAPH (MG)

A scholarly, professionally oriented publication dealing with state-of-the-art research or an authoritative treatment of a broad area. Expected to have long-lasting value.

NTIA SPECIAL PUBLICATION (SP)

Conference proceedings, bibliographies, selected speeches, course and instructional materials, directories, and major studies mandated by Congress.

NTIA REPORT (TR)

Important contributions to existing knowledge of less breadth than a monograph, such as results of completed projects and major activities. Subsets of this series include:

NTIA RESTRICTED REPORT (RR)

Contributions that are limited in distribution because of national security classification or Departmental constraints.

NTIA CONTRACTOR REPORT (CR)

Information generated under an NTIA contract or grant, written by the contractor, and considered an important contribution to existing knowledge.

JOINT NTIA/OTHER-AGENCY REPORT (JR)

This report receives both local NTIA and other agency review. Both agencies' logos and report series numbering appear on the cover.

NTIA SOFTWARE & DATA PRODUCTS (SD)

Software such as programs, test data, and sound/video files. This series can be used to transfer technology to U.S. industry.

NTIA HANDBOOK (HB)

Information pertaining to technical procedures, reference and data guides, and formal user's manuals that are expected to be pertinent for a long time.

NTIA TECHNICAL MEMORANDUM (TM)

Technical information typically of less breadth than an NTIA Report. The series includes data, preliminary project results, and information for a specific, limited audience.

For information about NTIA publications, contact the NTIA/ITS Technical Publications Office at 325 Broadway, Boulder, CO, 80305 Tel. (303) 497-3572 or e-mail info@its.bldrdoc.gov.

This report is for sale by the National Technical Information Service, 5285 Port Royal Road, Springfield, VA 22161,Tel. (800) 553-6847.