NTIA Technical Memorandum TM-11-475

Variable Frame Delay (VFD) Parameters for Video Quality Measurements

Stephen Wolf

NTIA Technical Memorandum TM-11-475

Variable Frame Delay (VFD) Parameters for Video Quality Measurements

Stephen Wolf

U.S. DEPARTMENT OF COMMERCE
April 2011

DISCLAIMER

CONTENTS

VARIABLE FRAME DELAY (VFD) PARAMETERS FOR VIDEO QUALITY MEASUREMENTS

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, after which the video may continue with or without skipping video frames. Sometimes sections of the original video stream may be missing entirely (skipping without pausing). Time varying delays of the output (or processed) video frames with respect to the input (i.e., the original or reference) video frames present significant challenges for Full Reference (FR) video quality measurement systems. Time alignment errors between the output video sequence and the input video sequence can produce measurement errors that greatly exceed the perceptual impact of these time varying video delays. This document proposes several objective video quality parameters that can be extracted from variable frame delay (VFD) information, demonstrates their correlation to subjective video quality, and shows how they can be utilized in an FR video quality measurement (VQM) system.

Key words: alignment; calibration; correlation; dropped; frames; Full Reference (FR); objective; parameters; pausing; quality; skipping; subjective; time; variable delay; video; measurement

1. INTRODUCTION

Digital video transmission systems normally consist of a video encoder, a digital transmission method (e.g., Internet Protocol—IP), and a video decoder. Video frames passing through these systems can be dropped and/or subject to variable time delays. As a result, the presentation of the video to the end user may contain unnatural or jerky motion, pauses or frame freezes, and fast forwards or missing segments. Reasons for this type of behavior include (1) the video encoder may intelligently decide to reduce the video frame transmission rate in order to save bits, (2) the video decoder may decide to freeze the last good video frame when the digital transmission is interrupted or when errors such as IP packet loss are detected, or (3) sections of the video stream may not be encoded or may be dropped altogether by the transmission channel. Whatever the reasons, output video frames from modern video compression and transmission systems can

[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.

contain significant time varying video delays and a Full Reference (FR) video quality measurement (VQM) system must properly deal with these idiosyncrasies.

Time varying delays of the output (i.e., processed) video frames with respect to the input (i.e., original, or reference) present a challenge for perception-based FR VQMs. This is because the time alignment errors between the output video sequence and the input video sequence can produce measurement errors that greatly exceed the perceptual impact of these time varying video delays. For example, a one-frame video freeze without skipping will result in either the prior or later output segment being shifted by one video frame with respect to the original reference segment. While this is barely noticeable to viewers, the commonly used Peak-Signal-to-Noise-Ratio (PSNR) measurement [1] will detect a large impairment for the output video segment that is off by one video frame with respect to the original video segment.

Reference [2] describes an FR technique for estimating variable frame delay (VFD) information. This document describes a technique that uses the one presented in [2] but goes further by exploring perception-based quality parameters that can be extracted from the VFD information (Section 2). Also examined is the correlation of these VFD parameters to subjective quality ratings (Section 3), and how they might be used to improve existing VQMs, such as those found in [1] and [3] (Section 4).

Figure 1 shows one possible method for using VFD information in an FR VQM system. The processed video is calibrated to remove gain and level offsets, spatial shifts and spatial scaling (if present). Next, the VFD information (i.e., the best matching original frame for every processed frame) is calculated and used to change the original video sequence so it matches the processed video sequence (i.e., VFD-matched original video). For instance, if the processed video sequence repeated every other frame, then the original sequence would match this behavior. The VFD information generated from this step, together with the calibrated processed video, and the VFD-matched original video would all be sent to the VQM system. The VQM system would compute VFD quality parameters that quantify the perceptual effects of variable frame delays (pauses, skips, etc.) and this information would then be combined with the full reference quality measurement (with variable video delays removed) to produce improved estimates of overall video quality.

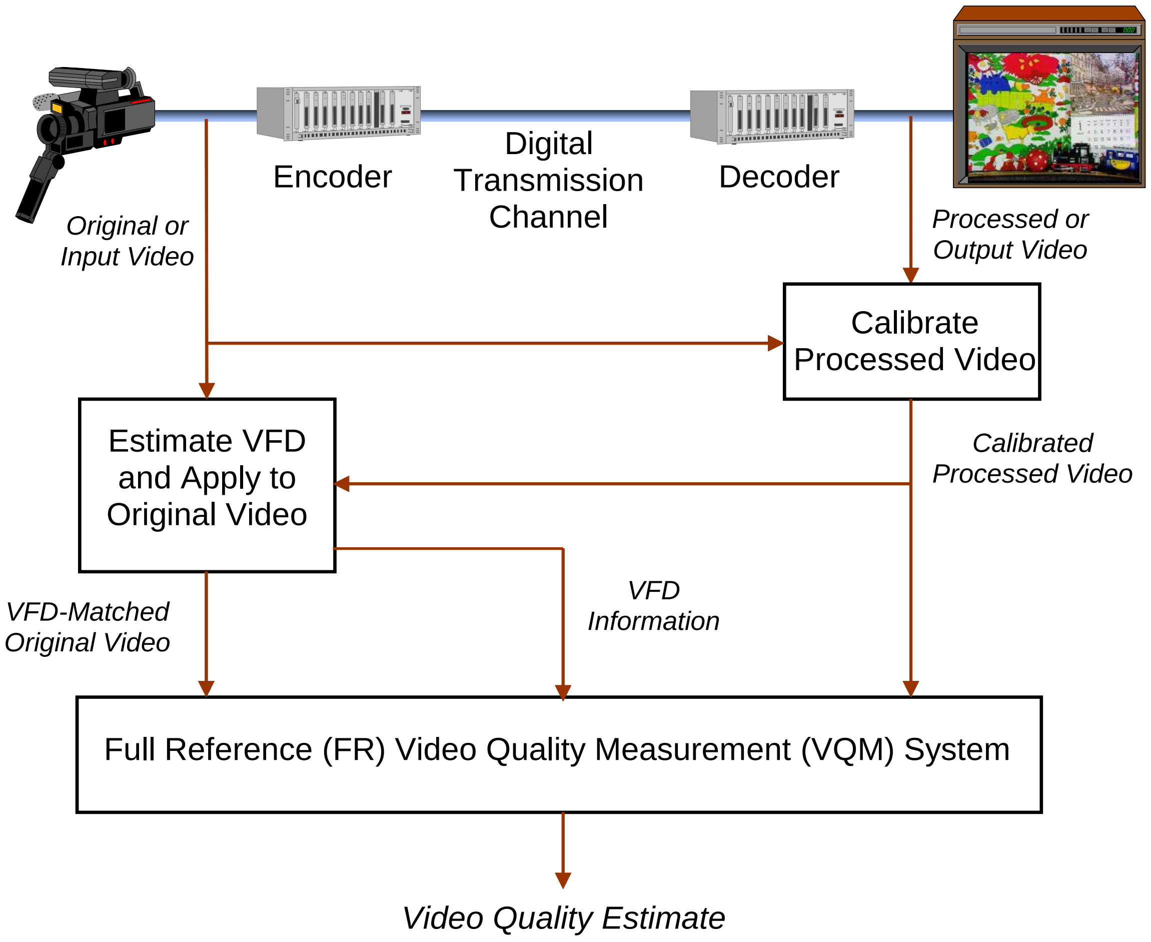
Encoder
Digital Transmission Channel
Decoder
Original or Input Video
Processed or Output Video
Calibrate Processed Video
Calibrated Processed Video
Estimate VFD and Apply to Original Video
VFD-Matched Original Video
VFD Information
Full Reference (FR) Video Quality Measurement (VQM) System
Video Quality Estimate

Figure 1. Schematic drawing of an FR VQM System with VFD capability.

2. DESCRIPTION OF VFD PARAMETERS

This section describes two objective quality parameters that can be derived from the VFD information. One parameter uses the VFD information exclusively (Section 2.1) and the other uses the VFD information in conjunction with the scene motion (Section 2.2). Both parameters were empirically developed with the goal of maximizing their correlation to the subjective quality ratings of short (8-15 second long) video test scenes (see Section 3).[2]

The technique described in this document uses the VFD algorithm given in [2] for determining the best matching original video frame[3] for each processed video frame. In particular, the technique uses the *Final_Fuzzy_Causal_Index* array generated by step 12, Section 2.6, of the VFD estimation algorithm. The *Final_Fuzzy_Causal_Index* is a 2D matrix, where the number of columns is equal to the number of frames in the processed clip, and the number of rows is equal to the number of original frames that were searched to try to match (or align) each processed frame. Each column of the 2D matrix gives a set of rank sorted original frame alignments, sorted from most likely to least likely. Thus, the first element of each column vector gives the most likely matching original frame index for that processed frame. Results for each processed frame only include likely alignments (i.e., original frame indices that closely match the processed frame).[4] For brevity, the 2D array *Final_Fuzzy_Causal_Index* for one processed video clip will simply be called *Fuzzy* in this document since it holds a set of fuzzy time alignments. The 2D *Fuzzy* array is denoted as:

$$Fuzzy(o_p, p): \quad p = 1,2,...,N; \quad o_p \in \left\{o_p^1, o_p^2, ..., o_p^{L_p}\right\}. \qquad (1)$$

In (1), p = 1, 2, ..., N (the total number of frames in the processed video clip) and each processed frame p matches a set o_p of L_p original frames, where L_p is the number of likely alignments for processed frame p. When the match is non-ambiguous, $L_p = 1$. When the match is ambiguous, $L_p > 1$. Video frames in processed scenes that contain very little motion, high distortions, and/or contributions from multiple original frames will create temporal alignment ambiguities.

2.1. Parameter 1 (Par1) Using VFD Information

This section describes an objective video quality parameter that is extracted solely from the VFD information. The idea is to penalize abnormal frame jumps in the VFD-matching original frame indices o_p in (1). An abnormal frame jump would be a jump forward by more than one frame as

[2] The Appendix provides MATLAB code that implements these parameters.

[3] For progressive scan video systems, the VFD algorithm uses frames. For interlaced video systems, the VFD algorithm uses fields. Some interlaced video systems reframe the output video and this complicates the VFD time alignment algorithm considerably. For a definition and explanation of reframing, please see Section 3.1.2 of [3]. For simplicity, this document will generally use the term “frame” or “frames” to describe the algorithm.

[4] In the MATLAB code given in the Appendix, the row entries that do not contain likely alignments are assigned ‘NaN’ (Not-a-Number) so that all columns in the 2D matrix have the same number of rows.

the processed frames *p* advance from 1 to *N*. One could use $\{o_p^1\}$ for this calculation (the index of the most likely matched original frame), but this would ignore the uncertainty of the estimated temporal alignments. This uncertainty can be accommodated by calculating the minimum required frame jump that satisfies the fuzzy temporal alignment of (1). For a given processed frame *p*, first calculate *Fuzzy_Max_Early*(*p*) , the maximum original frame index at time *p*, as:

$$Fuzzy\_Max\_Early(p) = \min\left(\max\left\{o_p^1, o_p^2, ..., o_p^{L_p}\right\}, o_{p+1}^1\right):\ p = 1,2,...,N-1. \qquad (2)$$

The minimum in (2) is used to limit the maximum original frame index at time *p* to be less than or equal to the most likely original frame index at time *p*+1 (for causality). In other words, the uncertainty in the original frame alignments for processed frame *p* is not allowed to exceed the most likely original frame alignment for processed frame *p*+1. Next, for a given processed frame *p*+1, calculate *Fuzzy_Min_Late*(*p*+1), the minimum original frame index at time *p*+1, as:

$$Fuzzy\_Min\_Late(p+1) = \max\left(\min\left\{o_{p+1}^1, o_{p+1}^2, ..., o_{p+1}^{L_{p+1}}\right\}, Fuzzy\_Max\_Early(p)\right). \qquad (3)$$

The maximum in (3) is used to limit the minimum original frame index at time *p*+1 to be greater than or equal to the *Fuzzy_Max_Early*(*p*). Next, one can calculate the Abnormal Frame Jumps (*AFJ*) at time *p*+1 as a function of the difference between (3) and (2):

$$AFJ\,(p+1) = \max\left(0, [Fuzzy\_Min\_Late(p+1) - Fuzzy\_Max\_Early(p) - 1]\right). \qquad (4)$$

In (4), one is subtracted from the difference between (3) and (2) before maxing with zero since this indicates a normal progression of video frames over time. Note that (4) does not penalize for frame freezes. Instead, penalties are incurred when the processed frame is updated to a new frame, which is perceived as an abnormal jump in the scene motion. Longer frame freezes produce larger values of *AFJ* when the processed frame is finally updated.

The video quality parameter *Par*1 is then computed as the logarithm of the root mean square of the *AFJ*(*p*+1) time samples (*p* = 1, 2, ..., N-1), or

$$Par1 = \log_{10}\left(1 + \sqrt{\text{mean}\left(AFJ^{\,2}\right)}\right). \qquad (5)$$

$AFJ^{\,2}$ represents an element-by-element square of the time series given by (4). One is added to the root mean square value to prevent the logarithm of zero, which also has the effect of limiting the lower bound (or no impairment condition) of the parameter to zero. Large abnormal frame jumps that result in long periods of video freezes are more heavily penalized than many small abnormal frame jumps since squaring large values has a proportionally greater effect on the overall mean.

2.2. Parameter 2 (Par2) Using VFD and Motion Information

The parameter described in section 2.1 only uses the VFD information. Thus, dropped or frozen frames will yield the same parameter value, or impairment, regardless of the amount of scene motion. For example, 10 frames per second (fps) transmission of a 30 fps original video scene (e.g., with 1 frame out of every 3 being transmitted) will penalize low and high motion scenes equally. An improvement to *Par*1 might incorporate motion-based weighting so that scenes with more motion produce higher parameter values than scenes with less motion. In the limit, still or nearly still scenes should be penalized only slightly when frames are dropped since these dropped frames are not very noticeable. One possible temporal information (*TI*) weighting function at time *p*+1 is computed as:

$$TI(p+1) = \log_{10}\left(1 + \sqrt{\underset{over\, i,j}{\text{mean}}\left\{\left[Y_{p+1}(i,j) - Y_p(i,j,)\right]^2\right\}}\right), \quad (i,j) \in SROI \,. \qquad (6)$$

In (6), one is added to the root mean square of the difference between the processed images at time *p*+1 and time *p* (e.g., one could use the Y channel in an ITU-R Recommendation BT.601 sampled video stream [4]), and the logarithm is computed. Calculations are performed using active pixels (*i*, *j*), or pixels that contain actual video information, within the Spatial Region of Interest (*SROI*). The advantages of (6) include a zero weighting function for still video frames (since the logarithm of one is zero) and compression of the large dynamic range that results from the root mean square operation.

An alternative form of (6) that is not shown is to compute the processed frame-by-frame differences but to perform the root mean square operation over a 3-dimensional segment of video that includes some small time extent (e.g., 0.2 seconds) about the processed frame of interest. Smoothing the motion information will reduce the impact of temporal alignment errors between the *TI* motion weighting function and the *AFJ* function given by (4), which is based on fuzzy temporal alignments.

A video quality parameter *Par*2 that uses both VFD and motion information can be computed as:

$$Par2 = \log_{10}\left(1 + \sqrt{\text{mean}\left(\left[AFJ \cdot TI\right]^2\right)}\right). \qquad (7)$$

$[AFJ \cdot TI]^2$ represents an element-by-element multiply and square of the two time series given by (4) and (6). As in (5), one is added to the root mean square value to prevent the logarithm of zero, which also has the effect of limiting the lower bound (or no impairment condition) of the parameter to zero.

3. CORRELATION OF VFD PARAMETERS

This section examines the correlation to subjective quality of *Par*1 and *Par*2 given in sections 2.1 and 2.2, respectively. The goal of this section is to assess the proportion of Mean Opinion Score (MOS) variance that can be explained by *Par*1 and *Par*2. Since *Par*1 and *Par*2 are pure temporal distortion parameters and do not measure spatial or color distortions, one would not expect them to explain a large proportion of the subjective MOS variance. Three datasets were examined, each containing multimedia video clips sent over Internet Protocol (IP) at one of three resolutions: Quarter Common Intermediate Format (QCIF), Common Intermediate Format (CIF), and Video Graphics Array (VGA). These datasets included video clips with variable frame delays, the subject of this document.

The QCIF, CIF, and VGA datasets included 1964, 2143, and 1766 video clips, respectively. Each dataset is a collection of 14 to 17 subjective tests that were performed for a variety of purposes over the past decade. The MOS from the individual subjective tests within each resolution were mapped to the [0, 1] common scale using the techniques in [5]. For the [0, 1] common scale, "0" is the no impairment condition (i.e., best quality) and "1" is the maximum impairment condition (i.e., worst quality).

Scatter plots of the [0, 1] common scale subjective quality versus *Par*1 and *Par*2 are shown in Figure 2. The top, middle, and bottom rows of the figure give the results for QCIF, CIF, and VGA, respectively. The left and right columns give the results for *Par*1 and *Par*2, respectively. For the *Par*2 results, 0.2 seconds of smoothing was used for *TI* as described in section 2.2. The Pearson correlation coefficients (ρ) are given in the titles of the individual graphs. Since the square of the Pearson correlation coefficient is the percent of the variance that is explained by the parameters, *Par*1 explains 30.8%, 16.3%, and 20.6% of the subjective variance in the QCIF, CIF, and VGA datasets, respectively. *Par*2 explains 38.1%, 20.2%, and 23.9% of the subjective variance in the QCIF, CIF, and VGA datasets, respectively. The high variance for the QCIF dataset is somewhat surprising considering that these parameters are purely temporal in nature and do not measure spatial or other distortions. One possible explanation for this observation is a dependency of spatial distortions on temporal distortions. For instance, lower frame rate video systems (which are penalized by *Par*1 and *Par*2) may also generally exhibit higher spatial distortions.

Both objective parameters have a fairly linear response to changes in subjective quality. *Par*1 exhibits some quantization banding that results from fixed frame rate systems (e.g., 15 fps, 10 fps, 5 fps). These quantization banding effects are less pronounced with *Par*2. The correlation results for *Par*2 are slightly better than the correlation results for *Par*1 in all three datasets. However, this does not necessarily imply that *Par*2 would be the better complement to an existing set of objective metrics that compose a perceptual video quality model. Both parameters (*Par*1 and *Par*2) tend to err on the side of under-detecting perceived impairments rather than over-detecting perceived impairments (i.e., the scatter plots have an upper triangular shape). Since these parameters only measure temporal distortions and not spatial distortions, one would expect this type of behavior.

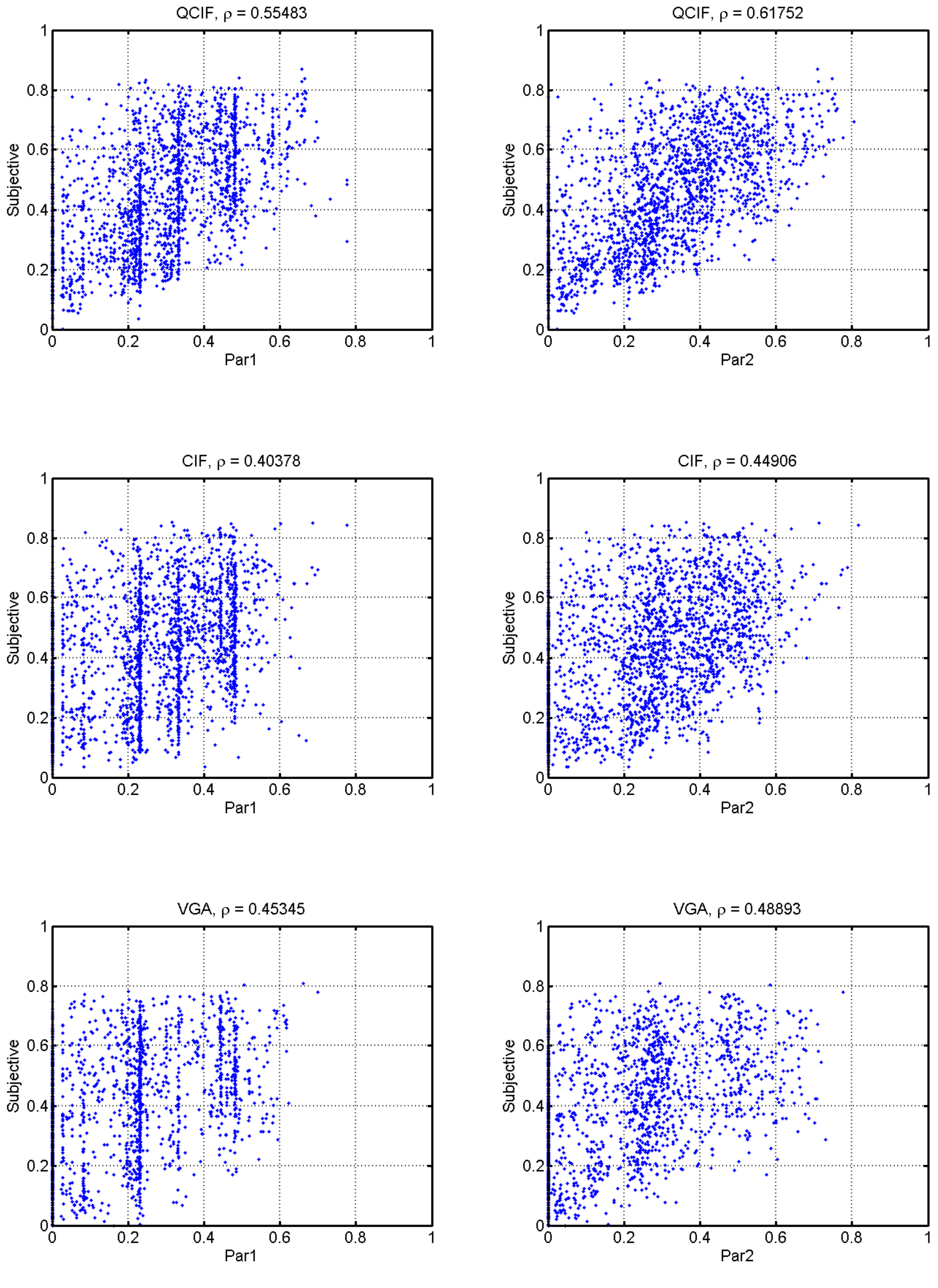

Figure 2. Subjective vs. objective plots for *Par*1 and *Par*2.

4. USING VFD PARAMETERS TO IMPROVE EXISTING VQMS

This section will examine how VFD processing can be used to improve the performance of FR VQMs, as summarized by the schematic in Figure 1. In particular, the performance of the PSNR metric given in [1] will be compared with the performance of a two-parameter VQM comprised of a linear combination of *Par*1 and the VFD-matched PSNR (PSNR_VFD). PSNR_VFD is computed as shown in Figure 1, where the original video clip is VFD-matched to the processed video clip before calculating PSNR. In essence, the PSNR of [1] includes errors due to both spatial distortions and temporal mis-alignments while PSNR_VFD does not include errors due to temporal mis-alignments of the video frames.[5] Thus, PSNR_VFD will primarily measure spatial distortions while *Par*1 will pick up the temporal distortions.

For the analysis in this section, subsets of the data from each of the three image resolutions (QCIF, CIF, VGA) evaluated by the Video Quality Experts Group (VQEG) Multimedia Test [6], [7] were selected. Individual subjective tests at each resolution were selected based on open availability of original scene content (i.e., no proprietary scenes), challenging content (e.g., poor performance from PSNR and/or the top video quality models), and the range of video quality. The chosen tests were (Q02, Q08, Q11), (C03, C07, C11), and (V02, V09, V12) for QCIF, CIF, and VGA, respectively. Including common video clips throughout the QCIF, CIF, and VGA subjective tests enabled the datasets at each resolution to be combined [7]. Hence, we will only present three sets of results (for QCIF, CIF, and VGA), rather than 9 sets of results (for Q02, Q08, Q11, C03, C07, C11, V02, V09, and V12). The subjective MOS for all datasets has been scaled to the [0, 1] common scale described earlier.

The PSNR reference code available at http://www.its.bldrdoc.gov/vqm/ was used for the computation of PSNR according to [1]. This reference code performs an exhaustive search over spatial and temporal shifts to locate the maximum PSNR for each video clip. This reference code processes all the video clip pairs (original, processed) in a given directory and outputs a PSNR results file with the maximum PSNR, together with the spatial shift, temporal shift, gain, and level offset that is required to obtain this maximum PSNR.

The Appendix provides MATLAB® code that implements PSNR_VFD, *Par*1, and *Par*2. To assist the reader, the variable names used in the MATLAB code are the same as those used in the main document. The PSNR_VFD code takes as its starting point the PSNR results file that is produced by the PSNR reference code. The spatial shift, gain, and level offset are assumed to be correct, and the temporal shift is used as a starting point for the VFD search algorithm described in [2]. After finding the original frame that best matches each processed frame, the original video clip is modified to VFD-match the processed video clip (see Figure 1). Gain and level offset are re-optimized to maximize PSNR_VFD. Next, PSNR_VFD, *Par*1, and *Par*2 are calculated and output to a file.

[5] In practice, the VFD matching algorithm is not perfect and this will result in some frames being temporally mis-aligned.

Figure 3 presents the QCIF performance for three video quality models. The first model (top-left) is a plot of the subjective vs. objective scores for PSNR as computed by the PSNR reference code. Here PSNR has been limited on the high end to 48 dB, which is a practical upper limit for video that is quantized to 8 bits.[6] Also, since PSNR does not normally have a linear response to changes in subjective quality, a third order monotonic fit has been performed to fit the PSNR values to the subjective data.[7] Similarly, the second model (top-right) presents the results for PSNR_VFD. Notice the improvement in Pearson correlation from 0.59 to 0.71. The third model (bottom) presents the results for a linear two-parameter model (PSNR_VFD & *Par*1), followed by a third order monotonic fit. Here, the correlation increases to 0.84.

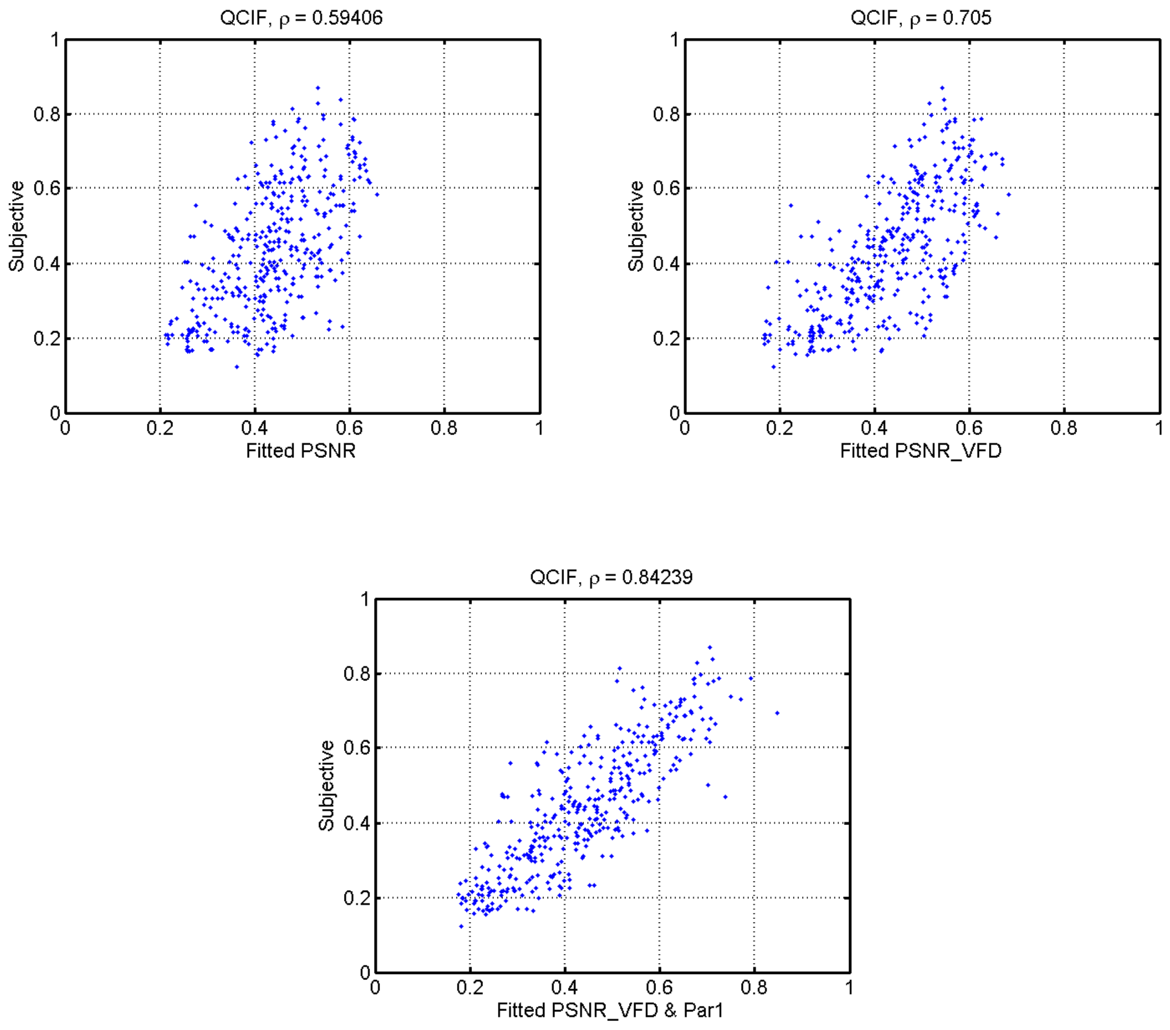

Figure 3. QCIF subjective vs. objective plots for PSNR, PSNR_VFD & *Par*1

[6] Other reasons to limit the upper end of the PSNR calculations include (1) *digital* video clips that are distortion free or nearly distortion free produce infinite or very large PSNR values which adversely impact the calculation of correlation coefficients and curve fits, and (2) impairments in video clips with PSNR values greater than 48 dB are not perceptible.

[7] This type of fit is often used by VQEG before evaluating the performance of an objective video quality model [6].

Figure 4 presents the analogous results for the CIF dataset. These results are similar to the QCIF results presented in Figure 3.

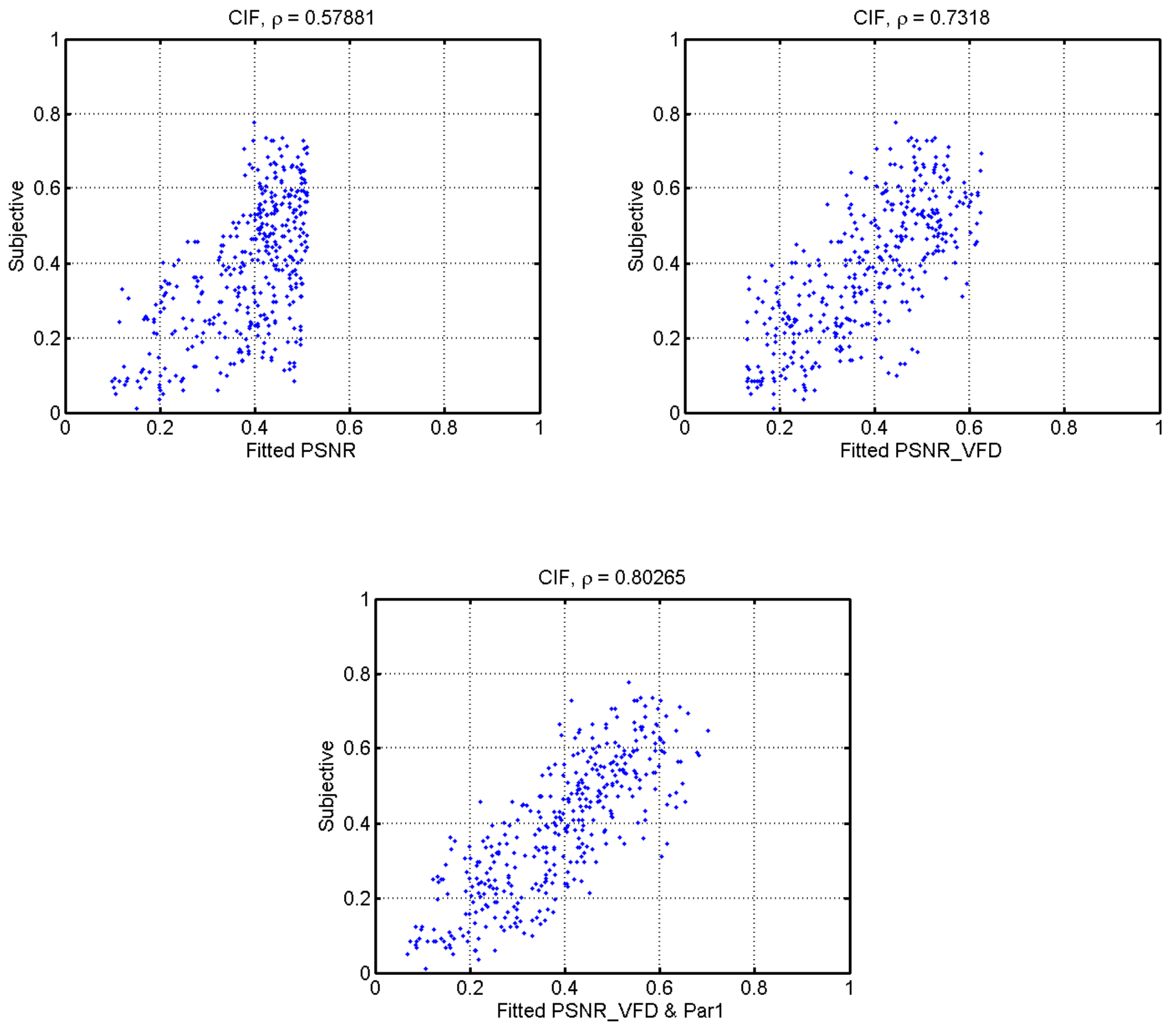

Figure 4. CIF subjective vs. objective plots for PSNR, PSNR_VFD & *Par*1

Figure 5 presents the corresponding results for the VGA dataset. Here the improvements in the PSNR_VFD and PSNR_VFD & *Par*1 models (vs. the PSNR model) are not nearly as dramatic as for the QCIF and CIF datasets. Possible reasons for this might be (1) a lower percentage of video clips with VFDs or reduced frame rates, (2) a reduced coupling of spatial distortions to temporal distortions, and (3) a greater contribution of spatial distortions to overall video quality that are not captured by PSNR.

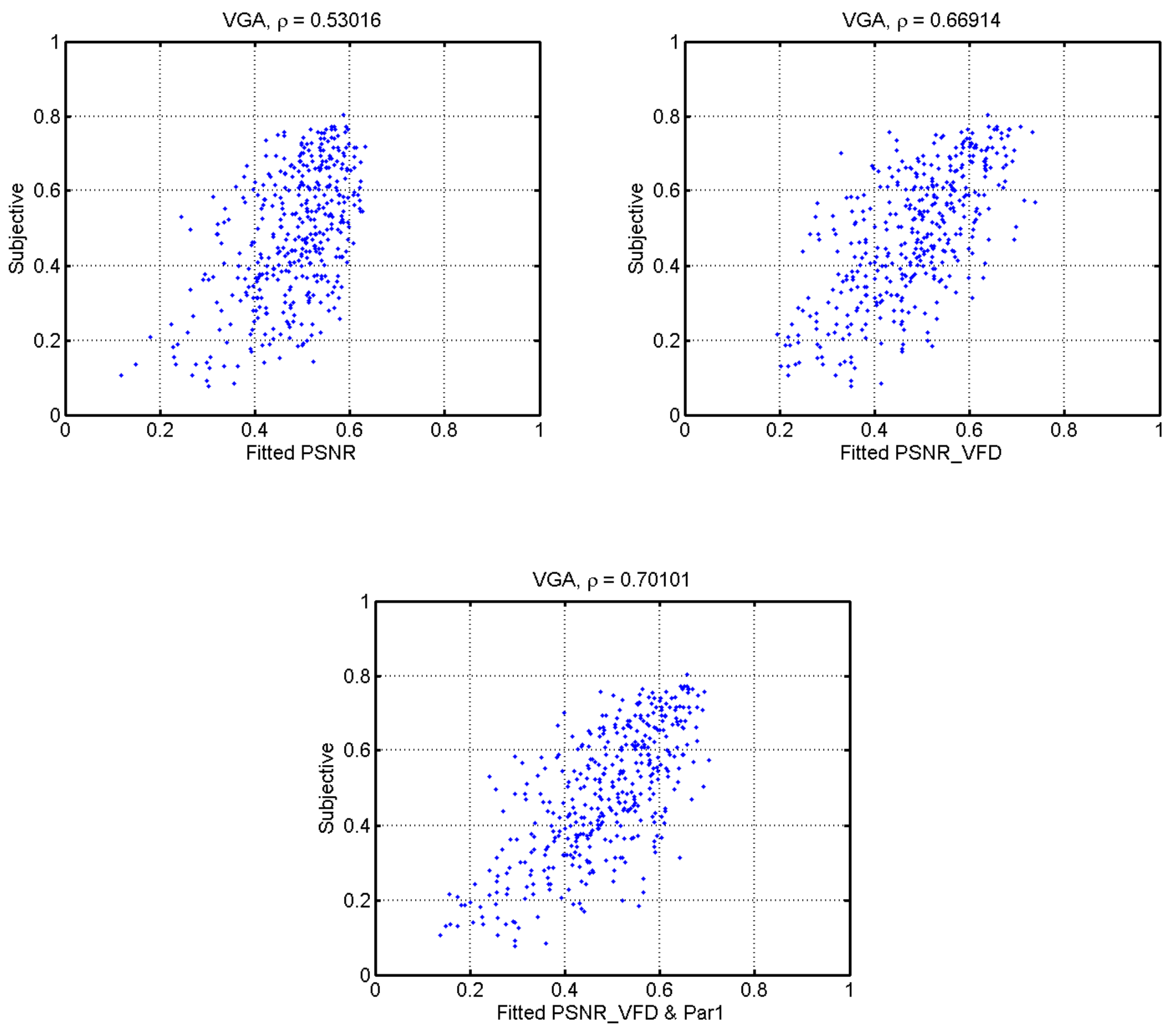

Figure 5. VGA subjective to objective plots for PSNR, PSNR_VFD & *Par*1

5. SUMMARY

This document describes two new video quality parameters that can be extracted from VFD information. One parameter uses pure VFD information (*Par*1) while the other uses VFD information weighted by the amount of scene motion (*Par*2). These VFD parameters explained from 16% to 38% of the MOS variance from subjective datasets at three different image resolutions (QCIF, CIF, and VGA). A procedure for designing a robust video quality measurement system that can properly handle VFD video clips was presented, and this procedure was applied to the PSNR measurement. Encouraging results were obtained that demonstrate improved performance (i.e., higher correlation to subjective quality). For one set of video clips drawn from the VQEG multimedia experiments (QCIF), correlation results improved from 0.59 for conventional PSNR to 0.84 for a two parameter model that used a linear combination of VFD-matched PSNR (PSNR_VFD) and a pure VFD parameter (*Par*1). Applying these methods to other video quality parameters that outperform the traditional PSNR should produce superior results, and this is an area for further research.

6. REFERENCES

[1] ITU-T Recommendation J.340, "Reference algorithm for computing peak signal to noise ratio of a processed video sequence with compensation for constant spatial shifts, constant temporal shift, and constant luminance gain and offset," Recommendations of the ITU, Telecommunication Standardization Sector.

[2] S. Wolf, "A full reference (FR) method using causality processing for estimating variable video delays," NTIA Technical Memorandum TM-10-463, Oct. 2009.

[3] S. Wolf, "Video quality measurement techniques," NTIA Technical Report 02-392, Jun. 2002.

[4] 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.

[5] M. Pinson, and S. Wolf, "An objective method for combining multiple subjective data sets," *SPIE Video Communications and Image Processing Conference*, Lugano, Switzerland, Jul. 2003.

[6] "Final Report of VQEG's Multimedia Phase I Validation Test", Video Quality Experts Group (VQEG), Sept. 2008. This report is available at http://www.its.bldrdoc.gov/vqeg/projects/multimedia/.

[7] M. Pinson, and S. Wolf, "Techniques for evaluating objective video quality models using overlapping subjective data sets," NTIA Technical Report TR-09-457, Nov. 2008.

APPENDIX: MATLAB Code

The PSNR_VFD function given below can be compiled (using the MATLAB compiler) and run from a DOS prompt. If compilation is performed, the routine is called as given in the help examples. For instance, the user would type the following at the DOS prompt:

psnr_vfd 'd:\q01\' 'q01' 'progressive' 'q01_psnr.csv' 'yuv' 144 176 'causal'

However, if psnr_vfd is run from the MATLAB prompt, the user would need to type the following at the MATLAB prompt:

psnr_vfd '''d:\q01\''' '''q01''' '''progressive ''' '''q01_psnr.csv''' '''yuv''' 144 176 '''causal'''

The PSNR_VFD function calls several other sub-functions that read video clips (i.e., read_bigyuv and read_avi). The MATLAB code for these sub-functions has not been included here since they are not required to understand the implementation of the basic PSNR_VFD algorithm. Code for these sub-functions are included in the PSNR_VFD source code software package which is available at http://www.its.bldrdoc.gov/vqm/.

```
function psnr_vfd(clip_dir, test, scan_type, psnr_file, varargin)
% PSNR_VFD 'clip_dir' 'test' 'scan_type' 'psnr_file' options
%   Estimate the Variable Frame Delay (VFD) Y-channel PSNR (PSNR) of all
%   clips and HRCs (Hypothetical Reference Circuits) in a video test (input
%   argument "test") where the video clips are stored in the specified
%   directory ("clip_dir").  The video clips must have names that conform to
%   the standard naming conventions test_scene_hrc.avi (or optionally,
%   test_scene_hrc.yuv) with no extra '_' or '.' in the file names.  "test"
%   is the name of the test, "scene" is the name of the scene, and "hrc" is
%   the name of the HRC.  The name of the original reference clip for the
%   PSNR calculation must be "test_scene_original.avi".
%
%   The user must specify the 'scan_type' of the video files as either
%   'progressive', 'interlaced_uff' (interlaced upper field first), or
%   'interlaced_lff' (interlaced lower field first), since this information
%   is not available in the AVI or the Big YUV file formats.
%
%   PSNR_VFD uses the results file output by the PSNR_SEARCH program
%   ("psnr_file") as a calibration starting point (i.e., Yshift, Xshift,
%   Tshift, Gain, Offset), but goes one step further by applying VFD
%   matching of the original video stream to the processed video stream
%   (i.e., the original video stream is modified so that it matches the
%   processed video stream frame-by-frame, or field-by-field for interlaced
%   video).  Output results from the PSNR_VFD program are stored in a file
%   with the same root name as "psnr_file" (i.e., name without the file
%   extension), appended with "_vfd.csv" (for Comma Separated Value).
%
% SYNTAX
%   psnr_vfd 'clip_dir' 'test' 'scan_type' 'psnr_file' options
%
% DESCRIPTION
%   This program uses the clip calibration information (i.e., Yshift,
%   Xshift, Tshift , Gain, Offset) from PSNR_SEARCH (in "psnr_file").  The
%   original clip is shifted by (Yshift, Xshift, Tshift) with respect to
%   the processed clip.  The Y-image of the processed clip is multiplied
%   by Gain and then the Offset is added.  For speed, all calibration is
%   held constant for the VFD estimation.  The VFD estimation algorithm is
%   applied to find the best matching original frame (or field) for each
%   processed frame (or field).  Then, the original is VFD-matched to the
%   processed.  A final gain and offset correction (called Gain_Adjust and
%   Offset_Adjust) is applied to the processed video (i.e.,
%   Gain_Adjust*y_proc + Offset_Adjust) before calculation of the final
%   VFD-corrected PSNR (PSNR_VFD).  Two perception-based VFD parameters are
%   also calculated (Par1_VFD and Par2_VFD) that attempt to capture the
%   perceptual distortions in the flow of scene motion (since these
%   distortions are removed from PSNR_VFD).  Par1_VFD is extracted from
%   only the VFD information while Par2_VFD uses both the VFD information
%   and the motion in the processed video clip.  For a description of these
%   two parameters, see the 2011 NTIA Technical Memorandum (TM) entitled
%   "Variable Frame Delay (VFD) Parameters for Video Quality Measurements."
%
%   The above procedure is repeated for each processed clip in the
%   "clip_dir" that belongs to the video test specified by "test".
%
%   A peak signal of 255 is used for calculation of PSNR.  Double precision
%   calculations are used everywhere.  A 64-bit operating system with at
%   least 4 GB of free memory is recommended since the entire double
%   precision versions of the original and processed sequences must be held
%   in memory.
%
%   Any or all of the following optional properties may be requested (the
%   first option is required for yuv files, not avi files).
%
%   'yuv' rows cols    Specifies the number of rows and cols for the Big
%                      YUV files (if using Big YUV files).  The default is
%                      to assume AVI files (*.avi).  Big YUV format is a
%                      binary format for storing ITU-R Recommendation
%                       BT.601 video sequences.  The format can be used for
%                       any image size.  In the Big YUV format, all the
%                       frames are stored sequentially in one big binary
%                       file. The sampling is 4:2:2 and image pixels are
%                       stored sequentially by video scan line as bytes in
%                       the following order: Cb1 Y1 Cr1 Y2 Cb3 Y3 Cr3 Y4…,
%                       where Y is the luminance component, Cb is the blue
%                       chrominance component, Cr is the red chrominance
%                       component, and the subscript is the pixel number.
%
%   'sroi' top left bottom right    Only use the specified spatial region
%                                   of interest (sroi) of the processed
%                                   video clip for all calculations. The
%                                   sroi is inclusive, where top/left start
%                                   at 1. By default, sroi is the entire
%                                   image reduced by the calibration shift
%                                   (Yshift, Xshift).  If the user inputs a
%                                   sroi, allowance must be made for the
%                                   maximum spatial shift encountered in
%                                   the PSNR_SEARCH results ("psnr_file").
%                                   For interlaced video, top must be odd
%                                   while bottom must be even.
%
%   'troi' fstart fstop    Only calculate PSNR_VFD for the specified
%                          temporal region of interest (troi) of the
%                          processed video clip, where fstart and fstop are
%                          included and given in frames.  By default, the
%                          troi is the entire file reduced by the temporal
%                          calibration shift (Tshift).  If the user inputs
%                          an fstart and fstop, allowance must be made for
%                          the maximum temporal shift encountered in
%                          "psnr_file".
%
%   't_uncert' t    Specifies the temporal uncertainty (plus or minus t
%                   frames) over which to perform the VFD search.  The
%                   processed remains fixed and the original is shifted.
%                   The center (zero shift) point for the temporal search
%                   assumes the temporal alignment given by Tshift in the
%                   "psnr_file" results from PSNR_SEARCH. By default,
%                   temporal uncertainty is set to 30 frames.  It can have
%                   a minimum value of 1 frame.  When the original cannot
%                   be shifted by the temporal uncertainty (e.g., perhaps
%                   near the ends of the sequence), the original will be
%                   shifted up to the maximum extent possible.  To
%                   accomodate a full temporal search at the beginning and
%                    end of the sequence, increase fstart and decrease fstop
%                    in the processed troi accordingly.
%
%   'reframe'  Allow for the possibility that the processed video clip has
%              been reframing.  This option is only valid for a scan_type
%              of 'interlaced_uff' or 'interlaced_lff'.  Reframing can vary
%              throughout the processed clip, although this should be rare.
%              This option will increase the runtime substantially since
%              extra spatial shifts must be examined, but it should be used
%              if there is any possibility of time varying reframing
%              existing in the processed video clip.  See Section 3.1.2 of
%              NTIA Report TR-02-392 for a definition of reframing.  For
%              constant reframing that was properly detected by the
%              PSNR_SEARCH program, Tshift will have a half frame (0.5)
%              Tshift.  This condition will be detected/corrected by
%              PSNR_VFD (the original video is reframed accordingly).
%
%   'causal'   Impose causality constraint so that later frames (fields) in
%              the processed clip cannot align to original frames (fields)
%              that are earlier in time than found for the proceeding
%              processed frames (fields).  For interlaced video, a
%              one-field jump back in time is allowed since this is
%              indicative of a frozen frame.  By default, causality is
%              turned off (yes, codecs can output non-causal sequences).
%              But specifying the causal option is usually recommended.
%
%   'verbose'   Display intermediate progress during processing.
%
% EXAMPLES
%   These three examples illustrate the Big YUV format for QCIF, CIF, and
%   VGA.
%
%   psnr_vfd 'd:\q01\' 'q01' 'progressive' 'q01_psnr.csv' 'yuv' 144 176 'causal'
%   psnr_vfd 'd:\c01\' 'c01' 'progressive' 'c01_psnr.csv' 'yuv' 288 352 'causal'
%   psnr_vfd 'd:\v01\' 'v01' 'progressive' 'v01_psnr.csv' 'yuv' 480 640 'causal'
%
%   This example illustrates uncompressed UYVY AVI format for 525-line video.
%
%   psnr_vfd 'd:\rr525\' 'rr525' 'interlaced_lff' 'rr525_psnr.csv' 'causal' 'reframe'
%

% This prints out help if the user runs with no command line arguments
if nargin == 0,
    fprintf('PSNR_VFD ''clip_dir'' ''test'' ''scan_type'' ''psnr_file'' options\n');
    fprintf('\n');
    fprintf('  Estimate the Variable Frame Delay (VFD) Y-channel PSNR (PSNR) of all\n');
    fprintf('  clips and HRCs (Hypothetical Reference Circuits) in a video test (input\n');
    fprintf('  argument "test") where the video clips are stored in the specified\n');
    fprintf('  directory ("clip_dir").  The video clips must have names that conform to\n');
    fprintf('  the standard naming conventions test_scene_hrc.avi (or optionally,\n');
    fprintf('  test_scene_hrc.yuv) with no extra ''_'' or ''.'' in the file names.  "test"\n');
    fprintf('  is the name of the test, "scene" is the name of the scene, and "hrc" is\n');
    fprintf('  the name of the HRC.  The name of the original reference clip for the\n');
    fprintf('  PSNR calculation must be "test_scene_original.avi".\n');
    fprintf('\n');
    fprintf('  The user must specify the ''scan_type'' of the video files as either\n');
    fprintf('  ''progressive'', ''interlaced_uff'' (interlaced upper field first), or\n');
    fprintf('  ''interlaced_lff'' (interlaced lower field first), since this information\n');
    fprintf('  is not available in the AVI or the Big YUV file formats.\n');
    fprintf('\n');
    fprintf('  PSNR_VFD uses the results file output by the PSNR_SEARCH program\n');
    fprintf('  ("psnr_file") as a calibration starting point (i.e., Yshift, Xshift,\n');
    fprintf('  Tshift, Gain, Offset), but goes one step further by applying VFD\n');
    fprintf('  matching of the original video stream to the processed video stream\n');
    fprintf('  (i.e., the original video stream is modified so that it matches the\n');
    fprintf('  processed video stream frame-by-frame, or field-by-field for interlaced\n');
    fprintf('  video).  Output results from the PSNR_VFD program are stored in a file\n');
    fprintf('  with the same root name as "psnr_file" (i.e., name without the file\n');
    fprintf('  extension), appended with "_vfd.csv" (for Comma Separated Value).\n');
    fprintf('\n');
    fprintf('SYNTAX\n');
    fprintf('  psnr_vfd ''clip_dir'' ''test'' ''scan_type'' ''psnr_file'' options\n');
    fprintf('\n');
    fprintf('DESCRIPTION\n');
    fprintf('  This program uses the clip calibration information (i.e., Yshift,\n');
    fprintf('  Xshift, Tshift , Gain, Offset) from PSNR_SEARCH (in "psnr_file").  The\n');
    fprintf('  original clip is shifted by (Yshift, Xshift, Tshift) with respect to\n');
    fprintf('  the processed clip.  The Y-image of the processed clip is multiplied\n');
    fprintf('  by Gain and then the Offset is added.  For speed, all calibration is\n');
    fprintf('  held constant for the VFD estimation.  The VFD estimation algorithm is\n');
    fprintf('  applied to find the best matching original frame (or field) for each\n');
    fprintf('  processed frame (or field).  Then, the original is VFD-matched to the\n');
    fprintf('  processed.  A final gain and offset correction (called Gain_Adjust and\n');
    fprintf('  Offset_Adjust) is applied to the processed video (i.e.,\n');
    fprintf('  Gain_Adjust*y_proc + Offset_Adjust) before calculation of the final\n');
    fprintf('  VFD-corrected PSNR (PSNR_VFD).  Two perception-based VFD parameters are\n');
    fprintf('  also calculated (Par1_VFD and Par2_VFD) that attempt to capture the\n');
    fprintf('  perceptual distortions in the flow of scene motion (since these\n');
    fprintf('  distortions are removed from PSNR_VFD).  Par1_VFD is extracted from\n');
    fprintf('  only the VFD information while Par2_VFD uses both the VFD information\n');
    fprintf('  and the motion in the processed video clip.  For a description of these\n');
    fprintf('  two parameters, see the 2011 NTIA Technical Memorandum (TM) entitled\n');
    fprintf('  "Variable Frame Delay (VFD) Parameters for Video Quality Measurements."\n');
    fprintf('\n');
    fprintf('  The above procedure is repeated for each processed clip in the\n');
    fprintf('  "clip_dir" that belongs to the video test specified by "test".\n');
    fprintf('\n');
    fprintf('  A peak signal of 255 is used for calculation of PSNR.  Double precision\n');
    fprintf('  calculations are used everywhere.  A 64-bit operating system with at\n');
    fprintf('  least 4 GB of free memory is recommended since the entire double\n');
    fprintf('  precision versions of the original and processed sequences must be held\n');
    fprintf('  in memory.\n');
    fprintf('\n');
    fprintf('  Any or all of the following optional properties may be requested (the\n');
    fprintf('  first option is required for yuv files, not avi files).\n');
    fprintf('\n');
    fprintf('  ''yuv'' rows cols    Specifies the number of rows and cols for the Big\n');
    fprintf('                     YUV files (if using Big YUV files).  The default is\n');
    fprintf('                     to assume AVI files (*.avi).  Big YUV format is a\n');
    fprintf('                     binary format for storing ITU-R Recommendation\n');
    fprintf('                     BT.601 video sequences.  The format can be used for\n');
    fprintf('                     any image size.  In the Big YUV format, all the\n');
    fprintf('                     frames are stored sequentially in one big binary\n');
    fprintf('                     file. The sampling is 4:2:2 and image pixels are\n');
    fprintf('                     stored sequentially by video scan line as bytes in\n');
    fprintf('                     the following order: Cb1 Y1 Cr1 Y2 Cb3 Y3 Cr3 Y4…,\n');
    fprintf('                     where Y is the luminance component, Cb is the blue\n');
    fprintf('                     chrominance component, Cr is the red chrominance\n');
    fprintf('                     component, and the subscript is the pixel number.\n');
    fprintf('\n');
    fprintf('  ''sroi'' top left bottom right    Only use the specified spatial region\n');
    fprintf('                                  of interest (sroi) of the processed\n');
    fprintf('                                  video clip for all calculations. The\n');
    fprintf('                                  sroi is inclusive, where top/left start\n');
    fprintf('                                  at 1. By default, sroi is the entire\n');
    fprintf('                                  image reduced by the calibration shift\n');
    fprintf('                                  (Yshift, Xshift).  If the user inputs a\n');
    fprintf('                                  sroi, allowance must be made for the\n');
    fprintf('                                  maximum spatial shift encountered in\n');
    fprintf('                                  the PSNR_SEARCH results ("psnr_file").\n');
    fprintf('                                  For interlaced video, top must be odd\n');
    fprintf('                                  while bottom must be even.\n');
    fprintf('\n');
    fprintf('  ''troi'' fstart fstop    Only calculate PSNR_VFD for the specified\n');
    fprintf('                         temporal region of interest (troi) of the\n');
    fprintf('                         processed video clip, where fstart and fstop are\n');
    fprintf('                         included and given in frames.  By default, the\n');
    fprintf('                         troi is the entire file reduced by the temporal\n');
    fprintf('                         calibration shift (Tshift).  If the user inputs\n');
    fprintf('                         an fstart and fstop, allowance must be made for\n');
    fprintf('                         the maximum temporal shift encountered in\n');
    fprintf('                         "psnr_file".\n');
    fprintf('\n');
    fprintf('  ''t_uncert'' t    Specifies the temporal uncertainty (plus or minus t\n');
    fprintf('                  frames) over which to perform the VFD search.  The\n');
    fprintf('                  processed remains fixed and the original is shifted.\n');
    fprintf('                  The center (zero shift) point for the temporal search\n');
    fprintf('                  assumes the temporal alignment given by Tshift in the\n');
    fprintf('                  "psnr_file" results from PSNR_SEARCH. By default,\n');
    fprintf('                  temporal uncertainty is set to 30 frames.  It can have\n');
    fprintf('                  a minimum value of 1 frame.  When the original cannot\n');
    fprintf('                  be shifted by the temporal uncertainty (e.g., perhaps\n');
    fprintf('                  near the ends of the sequence), the original will be\n');
    fprintf('                  shifted up to the maximum extent possible.  To\n');
    fprintf('                  accomodate a full temporal search at the beginning and\n');
    fprintf('                  end of the sequence, increase fstart and decrease fstop\n');
    fprintf('                  in the processed troi accordingly.\n');
    fprintf('\n');
    fprintf('  ''reframe''  Allow for the possibility that the processed video clip has\n');
    fprintf('             been reframing.  This option is only valid for a scan_type\n');
    fprintf('             of ''interlaced_uff'' or ''interlaced_lff''.  Reframing can vary\n');
    fprintf('             throughout the processed clip, although this should be rare.\n');
    fprintf('             This option will increase the runtime substantially since\n');
    fprintf('             extra spatial shifts must be examined, but it should be used\n');
    fprintf('             if there is any possibility of time varying reframing\n');
    fprintf('             existing in the processed video clip.  See Section 3.1.2 of\n');
    fprintf('             NTIA Report TR-02-392 for a definition of reframing.  For\n');
    fprintf('             constant reframing that was properly detected by the\n');
    fprintf('             PSNR_SEARCH program, Tshift will have a half frame (0.5)\n');
    fprintf('             Tshift.  This condition will be detected/corrected by\n');
    fprintf('             PSNR_VFD (the original video is reframed accordingly).\n');
    fprintf('\n');
    fprintf('  ''causal''   Impose causality constraint so that later frames (fields) in\n');
    fprintf('             the processed clip cannot align to original frames (fields)\n');
    fprintf('             that are earlier in time than found for the proceeding\n');
    fprintf('             processed frames (fields).  For interlaced video, a\n');
    fprintf('             one-field jump back in time is allowed since this is\n');
    fprintf('             indicative of a frozen frame.  By default, causality is\n');
    fprintf('             turned off (yes, codecs can output non-causal sequences).\n');
    fprintf('             But specifying the causal option is usually recommended.\n');
```

```
    fprintf('\n');
    fprintf('  ''verbose''   Display intermediate progress during processing.\n');
    fprintf('\n');
    fprintf('EXAMPLES\n');
    fprintf('  These three examples illustrate the Big YUV format for QCIF, CIF, and VGA\n');
    fprintf('\n');
    fprintf('  psnr_vfd ''d:\\q01\\'' ''q01'' ''progressive'' ''q01_psnr.csv'' ''yuv'' 144 176 ''causal''\n');
    fprintf('  psnr_vfd ''d:\\c01\\'' ''c01'' ''progressive'' ''c01_psnr.csv'' ''yuv'' 288 352 ''causal''\n');
    fprintf('  psnr_vfd ''d:\\v01\\'' ''v01'' ''progressive'' ''v01_psnr.csv'' ''yuv'' 480 640 ''causal''\n');
    fprintf('\n');
    fprintf('  This example illustrates uncompressed UYVY AVI format for 525-line video.\n');
    fprintf('\n');
    fprintf('  psnr_vfd ''d:\\rr525\\'' ''rr525'' ''interlaced_lff'' ''rr525_psnr.csv'' ''causal'' ''reframe''\n');
    fprintf('\n');
    return;
end

% Strip off the extra single quotes '' on the required inputs.  This is
% required for command line arguments in standalone execuatables.
clip_dir = eval(clip_dir);
test = eval(test);
scan_type = eval(scan_type);
psnr_file = eval(psnr_file);

% Generate the name of the file to store PSNR_VFD results
dot = strfind(psnr_file,'.');
vfd_file = strcat(psnr_file(1:dot(length(dot))-1), '_vfd.csv');

% Validate the scan_type
if (~strcmpi(scan_type,'progressive') && ~strcmpi(scan_type,'interlaced_lff') && ~strcmpi(scan_type,'interlaced_uff'))
    error('Invalid scan_type');
end

% Define the peak signal level
peak = 255.0;

% Define the sub-sampling factor on the pixels for the final gain and
% offset adjusting fit, which is performed right before calculation of
% PSNR_VFD.
fraction_sampled = 0.1;

% Add extra \ in clip_dir in case user did not
clip_dir = strcat(clip_dir,'\');

% Validate input arguments and set their defaults
```

```
file_type = 'avi';  % default file type, uncompressed UYVY AVI
is_yuv = 0;
is_whole_image = 1;
is_whole_time = 1;
t_uncert = 30;  % Default plus or minus temporal search, in frames
reframe = 0;
causal = 0;
verbose = 0;

cnt=1;
while cnt <= length(varargin),
    if strcmpi(eval(char(varargin(cnt))),'yuv') == 1
        rows = str2double(varargin{cnt+1});
        cols = str2double(varargin{cnt+2});
        is_yuv = 1;
        file_type = 'yuv';
        cnt = cnt + 3;
    elseif strcmpi(eval(char(varargin(cnt))),'sroi') == 1
        top = str2double(varargin{cnt+1});
        left = str2double(varargin{cnt+2});
        bottom = str2double(varargin{cnt+3});
        right = str2double(varargin{cnt+4});
        is_whole_image = 0;
        cnt = cnt + 5;
    elseif strcmpi(eval(char(varargin(cnt))),'troi') == 1
        fstart = str2double(varargin{cnt+1});
        fstop = str2double(varargin{cnt+2});
        is_whole_time = 0;
        cnt = cnt + 3;
    elseif strcmpi(eval(char(varargin(cnt))), 't_uncert') ==1
        t_uncert = str2double(varargin{cnt+1});
        cnt = cnt + 2;
    elseif strcmpi(eval(char(varargin(cnt))),'reframe') == 1
        reframe = 1;
        cnt = cnt + 1;
        % Make sure video is not progressive for this option
        if (strcmpi(scan_type,'progressive'))
            error('Reframe option not allowed for progressive video');
        end
    elseif strcmpi(eval(char(varargin(cnt))),'causal') == 1
        causal = 1;
        cnt = cnt + 1;
    elseif strcmpi(eval(char(varargin(cnt))),'verbose') == 1
        verbose = 1;
        cnt = cnt +1;
    else
        error('Property value passed into psnr_vfd not recognized');
    end
end

% If not progressive and user inputs an SROI, they must have an odd top and
% an even bottom.  Otherwise the field ordering will reverse.
if (~strcmpi(scan_type,'progressive') && ~is_whole_image && (~mod(top,2) || mod(bottom,2)))
    error('SROI top must be odd and bottom must be even for interlaced video.');
end

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

% Find the HRCs and their scenes for the specified video test
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,'.');  % Will only use the last dot
    if(size(und,2)==2) % possible standard naming convention file found
        this_test = this_file(1:und(1)-1);  % pick off the test name
        if(~isempty(strmatch(test,this_test,'exact')) && ...
                ~isempty(strmatch(file_type,this_file(dot(length(dot))+1:length(this_file)),'exact')))  % test clip found
            this_scene = this_file(und(1)+1:und(2)-1);
            this_hrc = this_file(und(2)+1:dot(length(dot))-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
                if(size(strmatch(this_scene,scene_list{loc},'exact'),1)==0)
                    scene_list{loc} = [scene_list{loc} this_scene];
                end
            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);
```

```
if (num_hrcs == 0)
    error('No files with standard naming convention found.');
end

% Results struct to store psnr_vfd results
results_vfd = struct('test', {}, 'scene', {}, 'hrc', {}, 'gain_adjust', {}, 'offset_adjust', {}, ...
    'psnr_vfd', {}, 'par1_vfd', {}, 'par2_vfd', {}, 'orig_indices', {}, 'proc_indices', {});

% Read in the psnr results output by the PSNR_SEARCH program.  Test clips
% that do not have PSNR_SEARCH results will be skipped.
psnr_import = importdata(psnr_file);
%  Check to make sure that the imported structure has the correct
%  characteristics for the textdata and data arrays
if (size(psnr_import.textdata,2)~=9 || size(psnr_import.data,2)~=6 || size(psnr_import.textdata,1)-1 ~= size(psnr_import.data,1))
    error('Invalid psnr_file.');
end
nclips = size(psnr_import.data,1);  % The number of clips in the file
% Load the structure to hold the PSNR_SEARCH results
results_psnr = struct('test', {}, 'scene', {}, 'hrc', {}, 'yshift', {}, ...
    'xshift', {}, 'tshift', {}, 'gain', {}, 'offset', {}, 'psnr', {});
for i = 1:nclips
    results_psnr(i).test = psnr_import.textdata{i+1,1};
    results_psnr(i).scene = psnr_import.textdata{i+1,2};
    results_psnr(i).hrc = psnr_import.textdata{i+1,3};
    results_psnr(i).yshift = psnr_import.data(i,1);
    results_psnr(i).xshift = psnr_import.data(i,2);
    results_psnr(i).tshift = psnr_import.data(i,3);
    results_psnr(i).gain = psnr_import.data(i,4);
    results_psnr(i).offset = psnr_import.data(i,5);
    results_psnr(i).psnr = psnr_import.data(i,6);
end

% Process one HRC at a time to compute average PSNR_VFD for that HRC
index = 1;  % index used to store psnr_vfd results
fid_vfd = fopen(vfd_file,'a');  % open vfd_file for appending and write out the header
if (strcmpi(scan_type,'progressive'))
    fprintf(fid_vfd,'Test,Scene,HRC,Gain_Adjust,Offset_Adjust,PSNR_VFD,Par1_VFD,Par2_VFD,(Proc Orig) Matching Frame Indices\n');
else % interlaced
    fprintf(fid_vfd,'Test,Scene,HRC,Gain_Adjust,Offset_Adjust,PSNR_VFD,Par1_VFD,Par2_VFD,(Proc Orig) Matching Field Indices\n');
end
fclose(fid_vfd);
for i = 1:num_hrcs

    psnr_ave = 0;  % psnr_vfd average summer for this HRC
    par1_ave = 0;  % par1_vfd average summer for this HRC
```

```
par2_ave = 0;  % par2_vfd average summer for this HRC
this_hrc = hrc_list{i};
if(strcmpi('original',this_hrc)) % Don't process original
    continue;
end
num_scenes = size(scene_list{i},2);  % Number of scenes in this HRC

for j = 1:num_scenes

    this_scene = scene_list{i}{j};

    %  Find this clip's calibration information in results_psnr
    this_clip  = find((strcmpi({results_psnr.test},test) & strcmpi({results_psnr.scene},this_scene) & strcmpi({results_psnr.hrc},this_hrc)));
    if (isempty(this_clip) && verbose)
        fprintf('Skipping Clip %s_%s_%s, Calibration information not found in psnr_file.\n', test, this_scene, this_hrc);
        continue;
    end

    %  Assign the scene information to the results_vfd structure
    results_vfd(index).test = test;
    results_vfd(index).scene = this_scene;
    results_vfd(index).hrc = this_hrc;

    %  Pick off the calibration information for this clip
    this_yshift = results_psnr(this_clip).yshift;
    this_xshift = results_psnr(this_clip).xshift;
    this_tshift = results_psnr(this_clip).tshift;
    this_gain = results_psnr(this_clip).gain;
    this_offset = results_psnr(this_clip).offset;
    this_psnr = results_psnr(this_clip).psnr;  % This will be used as a check against psnr_vfd

    % Read original and processed video files
    if (~is_yuv)  % YUV file parameters not specified, AVI assumed
        % Re-generate the original and processed avi file names
        orig = strcat(clip_dir, test,'_', this_scene, '_', 'original', '.avi');
        proc = strcat(clip_dir, test,'_', this_scene, '_', this_hrc, '.avi');
        [avi_info] = read_avi('Info',orig);
        [avi_info_proc] = read_avi('Info',proc);
        rows = avi_info_proc.Height;
        cols = avi_info_proc.Width;
        % Check to make sure processed and original sizes match
        rows_orig = avi_info.Height;
        cols_orig = avi_info.Width;
        if (rows ~= rows_orig || cols ~= cols_orig)
            error('Original and processed image sizes do not match.')
```

```
end

% Set/Validate the SROI of the processed video
if (is_whole_image) % make SROI the whole image less the calibration shift
    if (this_xshift <= 0)  % Original is shifted left or 0 wrt processed
        left = 1-this_xshift;
        right = cols;
    else  % Original is shifted right wrt processed
        left = 1;
        right = cols-this_xshift;
    end
    if (this_yshift <= 0)  % Original is shifted up or 0 wrt processed
        top = 1-this_yshift;
        if(~strcmpi(scan_type,'progressive') && ~mod(top,2))  % Must start on odd line for interlaced video
            top = top + 1;
        end
        bottom = rows;
    else  % Original is shifted down wrt processed
        top = 1;
        bottom = rows-this_yshift;
        if(~strcmpi(scan_type,'progressive') && mod(bottom,2))  % Must end on even line for interlaced video
            bottom = bottom - 1;
        end
    end
end
if (top<1 || left<1 || bottom>rows || right>cols)
    fprintf('Skipping Clip %s_%s_%s, invalid processed SROI, top=%f, left=%f, bottom=%f, right = %f.\n', ...
        test, this_scene, this_hrc, top, left, bottom, right);
    continue;
end

%  Set the matching original SROI and validate
left_orig = left + this_xshift;
right_orig = right + this_xshift;
top_orig = top + this_yshift;
bottom_orig = bottom + this_yshift;
% Odd y_shift, correct to preserve field ordering for interlaced, new top and bottom create two extra lines that
% will be eliminated in the reframe.
if (mod(this_yshift,2) && ~strcmpi(scan_type,'progressive'))
    top_orig = top_orig - 1;
    bottom_orig = bottom_orig + 1;
end
if (top_orig<1 || left_orig<1 || bottom_orig>rows || right_orig>cols)  % Original SROI wrt processed SROI
    fprintf('Skipping Clip %s_%s_%s, original xshift=%f and yshift=%f\n', ...
        test, this_scene, this_hrc, this_xshift, this_yshift);
    fprintf('produces invalid SROI top_orig=%f, left_orig=%f, bottom_orig=%f, right_orig=%f.\n', ...
        top_orig, left_orig, bottom_orig, right_orig);
    continue;
end

tframes = avi_info.NumFrames;  % total frames in orig file
tframes_proc = avi_info_proc.NumFrames;
% Validate that orig and proc have the same number of frames
if (tframes ~= tframes_proc)
    fprintf('\n%s_%s_%s: orig & proc files have different number of frames; longer file will be truncated.\n', ...
        test, this_scene, this_hrc);
    tframes = min(tframes,tframes_proc);
end

% Set/Validate the time segment of the processed video
if (is_whole_time) % use whole time segment less the calibration shift
    if (this_tshift <= 0)  % original is shifted left or 0 wrt processed
        fstart = ceil(1-this_tshift);
        fstop = tframes;
    else  % original is shifted right wrt processed
        fstart = 1;
        fstop = floor(tframes-this_tshift);
    end
end
if (fstart<1 || fstop>tframes)
    fprintf('Skipping Clip %s_%s_%s, invalid processed TROI, fstart=%f, fstop=%f.\n', ...
        test, this_scene, this_hrc, fstart, fstop);
    continue;
end

% Set the matching original fstart and fstop and validate.  The original will contain an extra frame
% when reframing is required.
fstart_orig = floor(fstart+this_tshift);
fstop_orig = ceil(fstop+this_tshift);
if (fstart_orig<1 || fstop_orig>tframes)  % Original TROI wrt Processed TROI
    fprintf('Skipping Clip %s_%s_%s, original tshift=%f produces\n', test, this_scene, this_hrc, this_tshift);
    fprintf('invalid original TROI: fstart_orig=%f, fstop_orig=%f.\n', fstart_orig, fstop_orig);
    continue;
end

%  Calculate to see how many extra original frames there are at
%  the beginning and end of the sequence for VFD calculations.
%  We would like at least t_uncert extra frames.
beg_extra = min(fstart_orig-1, t_uncert);
end_extra = min(tframes-fstop_orig, t_uncert);
```

```
    first_align = 1 + beg_extra;  % This original frame best aligns to the first frame in the processed TROI

    % Read in video and clear color planes to free up memory
    [y_orig,cb,cr] = read_avi('YCbCr',orig,'frames',fstart_orig-beg_extra,...
        fstop_orig+end_extra, 'sroi',top_orig,left_orig,...
        bottom_orig,right_orig);
    clear cb cr;
    [y_proc,cb,cr] = read_avi('YCbCr',proc,'frames',fstart,fstop,...
        'sroi',top,left,bottom,right);
    clear cb cr;

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

    % Set/Validate the SROI of the processed video
    if (is_whole_image) % make SROI the whole image less the calibration shift
        if (this_xshift <= 0)  % Original is shifted left or 0 wrt processed
            left = 1-this_xshift;
            right = cols;
        else  % Original is shifted right wrt processed
            left = 1;
            right = cols-this_xshift;
        end
        if (this_yshift <= 0)  % Original is shifted up or 0 wrt processed
            top = 1-this_yshift;
            if(~strcmpi(scan_type,'progressive') && ~mod(top,2))  % Must start on odd line for interlaced video
                top = top + 1;
            end
            bottom = rows;
        else  % Original is shifted down wrt processed
            top = 1;
            bottom = rows-this_yshift;
            if(~strcmpi(scan_type,'progressive') && mod(bottom,2))  % Must end on even line for interlaced video
                bottom = bottom - 1;
            end
        end
    end
    if (top<1 || left<1 || bottom>rows || right>cols)
        fprintf('Skipping Clip %s_%s_%s, invalid processed SROI, top=%f, left=%f, bottom=%f, right = %f.\n', ...
            test, this_scene, this_hrc, top, left, bottom, right);
        continue;
    end
```

```
%  Set the matching original SROI and validate
left_orig = left + this_xshift;
right_orig = right + this_xshift;
top_orig = top + this_yshift;
bottom_orig = bottom + this_yshift;
% Odd y_shift, correct to preserve field ordering for interlaced, new top and bottom create two extra lines that
% will be eliminated in the reframe.
if (mod(this_yshift,2) && ~strcmpi(scan_type,'progressive'))
    top_orig = top_orig - 1;
    bottom_orig = bottom_orig + 1;
end
if (top_orig<1 || left_orig<1 || bottom_orig>rows || right_orig>cols)  % Original SROI wrt processed SROI
    fprintf('Skipping Clip %s_%s_%s, original xshift=%f and yshift=%f\n', ...
        test, this_scene, this_hrc, this_xshift, this_yshift);
    fprintf('produces invalid SROI, top_orig=%f, left_orig=%f, bottom_orig=%f, right_orig=%f.\n', ...
        top_orig, left_orig, bottom_orig, right_orig);
    continue;
end

% Find the total frames of the input original file
[fid, message] = fopen(orig, 'r');
if fid == -1
    fprintf(message);
    error('Cannot open this clip''s bigyuv file, %s', orig);
end
% Find last frame.
fseek(fid,0, 'eof');
tframes = ftell(fid) / (2 * rows * cols);
fclose(fid);
% Find the total frames of the processed 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_proc = ftell(fid) / (2 * rows * cols);
fclose(fid);
% Validate that orig and proc have the same number of frames
if (tframes ~= tframes_proc && verbose)
    fprintf('\n%s_%s_%s: orig & proc files have different number of frames; longer file will be truncated.\n', ...
        test, this_scene, this_hrc);
    tframes = min(tframes,tframes_proc);
end
```

```
    % Set/Validate the time segment of the processed video
    if (is_whole_time) % use whole time segment less the calibration shift
        if (this_tshift <= 0)  % original is shifted left or 0 wrt processed
            fstart = ceil(1-this_tshift);
            fstop = tframes;
        else  % original is shifted right wrt processed
            fstart = 1;
            fstop = floor(tframes-this_tshift);
        end
    end
    if (fstart<1 || fstop>tframes)
        fprintf('Skipping Clip %s_%s_%s, invalid processed TROI, fstart=%f, fstop=%f.\n', ...
            test, this_scene, this_hrc, fstart, fstop);
        continue;
    end

    % Set the matching original fstart and fstop and validate.  The original will contain an extra frame
    % when reframing is required.
    fstart_orig = floor(fstart+this_tshift);
    fstop_orig = ceil(fstop+this_tshift);
    if (fstart_orig<1 || fstop_orig>tframes)  % Original TROI wrt Processed TROI
        fprintf('Skipping Clip %s_%s_%s, original tshift=%f produces\n', test, this_scene, this_hrc, this_tshift);
        fprintf('invalid original TROI: fstart_orig=%f, fstop_orig=%f.\n', fstart_orig, fstop_orig);
        continue;
    end

    %  Calculate to see how many extra original frames there are at
    %  the beginning and end of the sequence for VFD calculations.
    %  We would like at least t_uncert extra frames.
    beg_extra = min(fstart_orig-1, t_uncert);
    end_extra = min(tframes-fstop_orig, t_uncert);
    first_align = 1 + beg_extra;  % This original frame best aligns to the first frame in the processed TROI

    % Read in video and clear color planes to free up memory
    [y_orig,cb,cr] = read_bigyuv(orig,'frames',fstart_orig-beg_extra,...
        fstop_orig+end_extra,'size',rows,cols,'sroi',top_orig,...
        left_orig,bottom_orig,right_orig);
    clear cb cr;
    [y_proc,cb,cr] = read_bigyuv(proc,'frames',fstart,fstop,...
        'size',rows,cols,'sroi',top,left,bottom,right);
    clear cb cr;

end
```

```
%  Reframe the original if required
if (rem(this_tshift,1))  % Non-integer tshift
    y_orig = reframe_video(y_orig, scan_type);
end

% Convert everything to double precision before any calculations
% are performed.
y_orig = double(y_orig);
y_proc = double(y_proc);

%  Correct the processed for gain and offset as read in from the
%  psnr_file.
y_proc = this_gain*y_proc + this_offset;

%  Generate the call to the VFD estimation function
func_call = 'est_var_frame_delays(y_proc,y_orig,''normalize'',';  % use normalize option as it seems to work best
if (reframe)
    func_call = strcat(func_call,'''reframe'',');
end
if (causal)
    func_call = strcat(func_call,'''causal'',');
end
if (verbose)
    func_call = strcat(func_call,'''verbose'',');
end
if (strcmpi(scan_type,'interlaced_lff'))
    func_call = strcat(func_call,'''interlaced'',1,');
    first_align = 2*first_align-1;  % convert to fields for est_var_frame_delays
end
if (strcmpi(scan_type,'interlaced_uff'))
    func_call = strcat(func_call,'''interlaced'',2,');
    first_align = 2*first_align-1;  % convert to fields for est_var_frame_delays
end
func_call = strcat(func_call,'''first_align'',',num2str(first_align),',','''t_uncert'',',num2str(t_uncert),')');

% Call the est_var_frame_delays function to get VFD results
[results results_rmse results_fuzzy results_fuzzy_mse] = eval(func_call);

%  If the VFD alignment algorithm failed, then results==0.
%  In that case, use the time alignment given by the psnr_file and
%  generate psuedo results where the alignment just increases by
%  one frame (or field) at a time.
[nrows, ncols, nframes] = size(y_proc);
if (results == 0)
    vfd_failed = 1;  % Set a logical variable to record that the VFD algorithm failed.
    if (~strcmpi(scan_type,'progressive'))  % interlaced
        npts = nframes*2;
    else  % progressive
        npts = nframes;
    end
    results = first_align:first_align+npts-1;
    fprintf('WARNING: VFD algorithm failed for clip %s_%s_%s, using psnr_file time alignment.\n', ...
        test, this_scene, this_hrc);
else
    vfd_failed = 0;
    npts = length(results);
end

% This code translates the VFD results to use the orig and proc FILE indexing
if (strcmpi(scan_type,'progressive'))
    proc_indices = (fstart-1) + (1:length(results));
    orig_indices = (fstart_orig-beg_extra-1) + results;
else % interlaced
    proc_indices = 2*(fstart-1) + (1:length(results));
    orig_indices = 2*(fstart_orig-beg_extra-1) + results;
end
results_vfd(index).orig_indices = orig_indices;
results_vfd(index).proc_indices = proc_indices;

% Apply the VFD correction to the original clip to make it look
% like the processed clip.
y_orig = vfd_match(y_orig, scan_type, results);

% Reshape for gain/offset fit and PSNR calculation
y_proc = reshape(y_proc,nrows*ncols*nframes,1);
y_orig = reshape(y_orig,nrows*ncols*nframes,1);

% Perform the final gain and offset fit using randomly sub-sampled pixels
rand_nums = round(randperm((nrows*ncols*nframes))); %Randomizes numbers from 1 to nrows*ncols*nframes
this_fit = polyfit(y_proc(rand_nums(1:round(nrows*ncols*nframes*fraction_sampled))),...
                        y_orig(rand_nums(1:round(nrows*ncols*nframes*fraction_sampled))),1);
clear rand_nums;
results_vfd(index).gain_adjust = this_fit(1);
results_vfd(index).offset_adjust = this_fit(2);

%  Calculate the final PSNR_VFD
this_psnr_vfd = 10*(log10(peak*peak)-log10(sum(((this_fit(1)*y_proc+this_fit(2))-y_orig).^2)/(nrows*ncols*nframes)));
results_vfd(index).psnr_vfd = this_psnr_vfd;
clear y_orig;  % Done with y_orig
```

```
%  Print out a warning if the psnr_vfd is significantly lower than
%  the psnr, which indicates that the VFD alignment might be
%  suspect.  Here we are using a threshold of 1.5 db.
if (this_psnr-this_psnr_vfd > 1.5)
    fprintf('WARNING: VFD alignment results may be unreliable for clip %s_%s_%s, psnr_vfd=%5.4f, psnr=%5.4f.\n', ...
        test, this_scene, this_hrc, this_psnr_vfd, this_psnr);
end

%  Reshape y_proc to 3D and calculate TI_RMS
y_proc = reshape(y_proc,nrows,ncols,nframes);
y_proc = cat(3,y_proc(:,:,1),y_proc);  % Add extra frame at the beginning for TI calculation
y_proc = diff(y_proc,1,3);  % first order difference along 3rd dimension
y_proc = reshape(y_proc,nrows*ncols,nframes);
y_proc = y_proc.^2;
y_proc = sqrt(sum(y_proc)./(nrows*ncols));
if (~strcmpi(scan_type,'progressive'))  % Replicate every other sample for interlaced video
    y_proc = reshape(repmat(y_proc,2,1), 1, 2*nframes);
end

%  Calculate the diff of the VFD information, which forms the basis
%  for both par1_vfd and par2_vfd.  The VFD information is
%  converted to a vector that gives Abnormal Frame Jumps (AFJs).
% Subtracting 1 and maxing with zero produces a parameter that (1)
% does not penalize for normal field/frame delivery (where the VFD
% field/frame indices increase by one from one field/frame to the
% next), (2) does not penalize for frame/field repeats (e.g., where
% the VFD frame indices stay fixed from one frame to the next), and
% (3) does not penalize for interlaced frame repeats (where the VFD
% field indices jump back one in time from one field to the next).
% A non-impairment value of 0 is used for the first field/frame
% (which must be padded since it's diff is not available).
if (vfd_failed)

    % This equation assumes that both the early and late time sides
    % used for the frame jump estimates are absolutely correct
    % (i.e., no fuzzy alignments).  The fuzzy alignment information
    % is not available here as the VFD algorithm failed.  Abnormal
    % Frame Jumps (AFJ) is then calculated as:
    afj = max([0 abs(diff(results))-1], 0);

else

    % This code assumes fuzzy uncertainty on both the early and
    % late time sides when calculating frame jumps (the uncertainty
    % is given by results_fuzzy array).  Here, frame jumps are only
```

```
        % penalized when they are absolutely certain to be correct
        % (i.e., no fuzzy overlapping).
        fuzzy_max_early = min(max(results_fuzzy(:,1:npts-1)),results(2:npts));
        fuzzy_min_late = max(min(results_fuzzy(:,2:npts)),fuzzy_max_early);
        afj = max([0 fuzzy_min_late-fuzzy_max_early-1], 0);

    end

    %  Calculate the pure VFD parameter (par1_vfd) and the TI weighted
    %  variant VFD parameter (par2_vfd).
    par1_vfd = log10(sqrt(mean(afj.^2))+1);
    results_vfd(index).par1_vfd = par1_vfd;
    par2_vfd = log10(sqrt(mean((afj.*log10(1+y_proc)).^2))+1);
    results_vfd(index).par2_vfd = par2_vfd;

    %  Output the clip information, current time, and the psnr_vfd
    t = clock;
    if (verbose)
        fprintf('Clip %s_%s_%s at %d:%d, psnr_vfd = %5.4f, par1_vfd = %5.4f, par2_vfd = %5.4f\n', ...
            test, this_scene, this_hrc, t(4), t(5), this_psnr_vfd, par1_vfd, par2_vfd);
    end

    % Write out the psnr_vfd results for this clip into the vfd_file
    fid_vfd = fopen(vfd_file,'a');
    fprintf(fid_vfd,'%s, %s, %s, %5.4f, %5.4f, %5.4f, %5.4f, %5.4f,', results_vfd(index).test, results_vfd(index).scene, ...
        results_vfd(index).hrc, results_vfd(index).gain_adjust, results_vfd(index).offset_adjust, ...
        results_vfd(index).psnr_vfd, results_vfd(index).par1_vfd, results_vfd(index).par2_vfd);
    for k = 1:npts-1
        fprintf(fid_vfd,'%d %d,',proc_indices(k), orig_indices(k));
    end
    fprintf(fid_vfd,'%d %d\n',proc_indices(k+1), orig_indices(k+1));
    fclose(fid_vfd);

    %  Add to the HRC summer
    psnr_ave = psnr_ave + this_psnr_vfd;
    par1_ave = par1_ave + par1_vfd;
    par2_ave = par2_ave + par2_vfd;

    %  Increment the results_vfd counter
    index = index+1;

end

% Compute average psnr_vfd for this HRC
    psnr_ave = psnr_ave/(num_scenes);
```

```
        par1_ave = par1_ave/(num_scenes);
        par2_ave = par2_ave/(num_scenes);
        if(verbose)
            fprintf('\nHRC = %s, psnr_vfd_ave = %5.4f, par1_vfd_ave = %5.4f, par2_vfd_ave = %5.4f\n\n', ...
                this_hrc, psnr_ave, par1_ave, par2_ave);
        end

        pause(1);
        close all;  % closes open figures
        fclose('all');  % closes open files

end


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [yout] = reframe_video(yin, scan_type)
%  function [yout] = reframe_video(yin, scan_type)
%  This function reframes a 3D input video array yin, with a scan_type of
%  either 'interlaced_uff' or 'interlaced_lff', and produces a 3D output
%  video array yout.  yout will have one less frame than yin and have its
%  number of rows reduced by two lines.  The number of rows in yin must be
%  even and yin must contain at least 2 video frames.
%
[nrows, ncols, nframes] = size(yin);
if (mod(nrows,2) || nframes<2)
    error ('reframe_video function requires an even number of rows and at least 2 video frames');
end

% Split_into_fields
yin = reshape(yin, 2, nrows/2, ncols, nframes);
if (strcmpi(scan_type,'interlaced_lff'))
    late_field = squeeze(yin(1,2:nrows/2,:,1:nframes-1));
    early_field = squeeze(yin(2,1:nrows/2-1,:,2:nframes));
elseif (strcmpi(scan_type,'interlaced_uff'))
    early_field = squeeze(yin(1,2:nrows/2,:,2:nframes));
    late_field = squeeze(yin(2,1:nrows/2-1,:,1:nframes-1));
else
    error('Unsupported scan_type in function reframe_video');
end
clear yin;

% Reframe video and return:
% For interlaced_lff the lower_field is now the late_field and the
% upper_field is now the early_field.
% For interlaced_uff the lower_field is now the early_field and the
% upper_field is now the late_field.
if (strcmpi(scan_type,'interlaced_lff'))  %  Reframe code for interlaced_lff
    yout(1,:,:,:) = early_field;
    yout(2,:,:,:) = late_field;
else  %  Reframe code for interlaced_uff
    yout(1,:,:,:) = late_field;
    yout(2,:,:,:) = early_field;
end
clear late_field early_field;
yout = reshape(yout, nrows-2, ncols, nframes-1);

return

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [yout] = vfd_match(yin, scan_type, results)
%  function [yout] = vfd_match(yin, scan_type, results)
%  This function converts 3D input video array 'yin' into another 3D video
%  array 'yout' with frames and/or fields ordered according to the Variable
%  Frame Delay (VFD) 'results'.  The scan_type must be either
%  'progressive', 'interlaced_lff', or 'interlaced_uff'.
%

[nrows, ncols, nframes_orig] = size(yin);

if (strcmpi(scan_type,'progressive'))
    nframes = length(results);
    is_interlaced = 0;
elseif (strcmpi(scan_type, 'interlaced_lff'))
    nframes = length(results)/2;  % These are field results, so must half to get nframes
    is_interlaced = 1;
    field_first = 1;  % Use same field_first definition as est_var_frame_delays
elseif (strcmpi(scan_type, 'interlaced_uff'))
    nframes = length(results)/2;
    is_interlaced = 1;
    field_first = 2;
else
    error('Unsupported scan_type in function vfd_match.');
end

% yout will be the VFD-corrected original and will have the same number of
% frames as the processed clip.  Use the same 'single' or 'double' rule as
% read_tslice for the image precision of yout.
if (nrows > 650)
    yout = zeros(nrows,ncols,nframes,'single');
else
```

```
    yout = zeros(nrows,ncols,nframes,'double');
end

if(is_interlaced)

    for j = 1:nframes
        % Get matching original field for the early processed field
        orig_frame_num = ceil(results(2*j-1)/2);  % The frame number that contains the original field
        early_field = mod(results(2*j-1),2);  % =1 if early field, =0 if late field
        [yo1 yo2] = split_into_fields(squeeze(yin(:,:,orig_frame_num)));
        if (early_field)
            switch field_first
                case(1)
                    this_orig1 = yo1;
                case(2)
                    this_orig1 = yo2;
            end
        else % late field
            switch field_first
                case(1)
                    this_orig1 = yo2;
                case(2)
                    this_orig1 = yo1;
            end
        end
        % Get matching original field for the late processed field
        orig_frame_num = ceil(results(2*j)/2);  % The frame number that contains the original field
        early_field = mod(results(2*j),2);  % =1 if early field, =0 if late field
        [yo1 yo2] = split_into_fields(squeeze(yin(:,:,orig_frame_num)));
        if (early_field)
            switch field_first
                case(1)
                    this_orig2 = yo1;
                case(2)
                    this_orig2 = yo2;
            end
        else % late field
            switch field_first
                case(1)
                    this_orig2 = yo2;
                case(2)
                    this_orig2 = yo1;
            end
        end
        % Joint the two original fields into a frame
        switch field_first
            case(1)
                this_orig = join_into_frames(this_orig1,this_orig2);
            case(2)
                this_orig = join_into_frames(this_orig2,this_orig1);
        end
        yout(:,:,j) = this_orig;
    end
    clear this_orig this_orig1 this_orig2 yo1 yo2;

else  % progressive

    for j = 1:nframes
        yout(:,:,j) = yin(:,:,results(j));
    end

end

return
```

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

BIBLIOGRAPHIC DATA SHEET

1. PUBLICATION NO. TM-11-475	2. Government Accession No.	3. Recipient's Accession No.
4. TITLE AND SUBTITLE Variable Frame Delay (VFD) Parameters for Video Quality Measurements		5. Publication Date April 2011 6. Performing Organization NTIA/ITS.T
7. AUTHOR(S) Stephen Wolf		9. Project/Task/Work Unit No. 3141011
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, after which the video may continue with or without skipping video frames. Sometimes sections of the original video stream may be missing entirely (skipping without pausing). Time varying delays of the output (or processed) video frames with respect to the input (i.e., the original or reference) video frames present significant challenges for Full Reference (FR) video quality measurement systems. Time alignment errors between the output video sequence and the input video sequence can produce measurement errors that greatly exceed the perceptual impact of these time varying video delays. This document proposes several objective video quality parameters that can be extracted from variable frame delay (VFD) information, demonstrates their correlation to subjective video quality, and shows how they can be utilized in an FR video quality measurement (VQM) system.

16. Key Words
alignment; calibration; correlation; dropped; frames; Full Reference (FR); objective; parameters; pausing; quality; skipping; subjective; time; variable delay; video; measurement

17. AVAILABILITY STATEMENT ☒ UNLIMITED.	18. Security Class. (This report) Unclassified	20. Number of pages 47
	19. Security Class. (This page) Unclassified	21. Price:

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.