1 Scope

This document describes two equations: 1) the V-NIIRS interpretability estimation equation that predicts the potential of the video to complete the tasks described in ST 0901 Video-NIIRS [1]; 2) a video quality equation that predicts the overall appearance of the video.

The calculated interpretability and quality can be inserted into the metadata stream using the keys defined in MISB ST 1108 [5], “Video Interpretability and Quality Keys”.

The difference between task-based interpretability and appearance-base video quality is defined by ITU-T Recommendation P.912, "Subjective video quality assessment methods for recognition tasks," [17].

The rationale for introducing the two new equations is that well known video quality metrics such as PSNR and SSIM judge the relative quality performance of encoders/decoders. However, PSNR and SSIM either fail, or cannot be used for the following cases:

1. No uncompressed reference is available
2. Stabilization
3. Contrast enhancement
4. Sharpening
5. Super-resolution, upsampling, and downsampling cannot use SSIM or PSNR because of size mismatch
6. When an intelligence interpretability metric is called for
7. When a metric is needed that takes advantage of existing metadata to analyze camera settings and motion

The Video Interpretability and Quality Measurement and Prediction techniques address all of the above situations where conventional metrics either are not suitable, or fail, or cannot be computed.

General comments are provided for the situations where either full or partial reference is available. Partial reference systems may employ sub-image chips and / or features in the metadata per ST 1108. Downstream receiving stations can use the chip/feature metadata to compute the Interpretability and Quality with improved accuracy.
2 References

2.1 Normative References

The following references and the references contained therein are normative.

[1] MISB ST 0901 Video NIIRS, Feb 2014
[3] MISB ST 0404.1, Compression for Infrared Motion Imagery, Feb 2014

2.2 Informative References

https://ece.uwaterloo.ca/~z70wang/research/ssim/ssim_index.m

3 Acronyms

A-FPS Analysis Frames-Per-Second
FPS Frames-Per-Second
4 Definitions

Blind Metrics
Metrics measured without the benefit of a reference or known calibration target.

DMOS
Difference Mean Opinion Score: difference mean opinion score between a mean opinion score for a source video data and a mean opinion score for the processed video data.

Interpretability
Refers to the ability of the video to satisfy task-based criteria specified in the Video-NIIRS

SSIM
Structural Similarity Index: a full reference metric that measures the similarity between two images.

5 Revision History

<table>
<thead>
<tr>
<th>Revision</th>
<th>Date</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RP 1203.3</td>
<td>02/27/2014</td>
<td>Promoted to RP; Added EARS format; updated references</td>
</tr>
</tbody>
</table>

6 Introduction

This document describes prototype algorithmic capability for measuring Video Interpretability and Quality metrics. No effort has been made to optimize the algorithms for runtime efficiency. The algorithms have not been tested over the entire range of motion imagery conditions that can be encountered. Note: the term “video” is to be considered the same as “motion imagery” in this document.

Interpretability measures the potential of the video for intelligence task completion.

Quality measures the overall appearance of the video.

The Video-NIIRS (ST 0901[1]) has interpretability levels ranging from 3 to 11. Spatial resolution is expressed in Ground Sample Distance (GSD). Temporal resolution is measured by sample frame rate. There are seven orders-of-battle: 1) Air Forces; 2) Electronics; 3) Ground Forces; 4) Missile; 5) Naval; 6) Culture; 7) Security. Each order-of-battle has at least one task defined for each of the nine increasingly difficult levels. An example of the most difficult level for the Culture order-of-battle:
“Confirm the movement of - a pedestrian's hands and fingers - as they make change or sort coins - in a busy open market or square - (individual of average height & weight, sorting coins in a change purse or the palm of one hand)"

Achievement of this task requires spatial and temporal resolution. Contrast with the least difficult task:

“Track the movement of - an oversized container carrying flatbed railcar – in motion - at a rail yard - (140 ft. length flatbed rolling stock)

* Note – under-resolved smaller trucks or rolling stock may be evident based on behavior.”

Much less spatial and temporal resolution is required because the object is large and slow moving.

The Video Quality has five levels: 1. Bad; 2. Poor; 3. Fair; 4. Good; 5. Excellent.

Video Quality set to zero indicates a forced setting that corresponds to “Bad”.

Quality definitions:

**Excellent:** no observable artifacts or video defects over sustained viewing intervals (1-hour). Overall balance and clarity is exceptional.

**Good:** occasional artifacts are observed no more than once-per-minute. Edges and detail are consistently sharp and well-contrasted.

**Fair:** consistent smooth video. Blurring and / or noise are barely noticeable. Annoyances and artifacts occur no more than twice-per-minute.

**Poor:** annoyances are observed every few seconds, or sustained impairments, such as blur or noise cause a noticeable and pronounced degradation.

**Bad:** content is barely discernable due to near constant presence of impairments.

Interpretability and quality are functions of:

1) Object size and distance;
2) Lighting condition, sun angle;
3) Motion (both camera and object);
4) Scene complexity (number of similar objects, ...).
5) Task type (detection, recognition, identification, track);
6) Latency requirements (forensic or live);
7) Viewing angle;
8) Camera specifics (MTF, FPS, pixel count, ...);
9) Calibration (White balance, non-uniformity correction, bad pixel maps, ...);
10) Atmosphere effects (turbulence, fog, ..);
11) Optics (mis-focus, aberrations, ...);
12) Artifacts due to compression and transmission.
Section 7 describes the Video Interpretability equations.

Section 8 describes the Video Quality equations.

### 7 Normative: Video Interpretability Equation

The instantaneous interpretability estimate for the $k$th frame is,

$$I_k = 14 - \log_2 \left( \frac{GSD_k}{RER_k} \right) - \log_2 \left( \frac{1}{RER_k} \right) - \exp(0.5 \cdot (\text{PSNR}_c - \text{PSNR}_k)) - \Delta I_{\text{camera}} - \Delta I_{\text{contrast}} - \Delta I_{\text{movers}} - \Delta I_{\text{artifacts}}$$

**Equation 1**

The equation is a function of resolution ($GSD$), blur ($RER$), noise ($\text{PSNR}$ dB), camera/platform motion ($\Delta I_{\text{camera}}$), overall contrast ($\Delta I_{\text{contrast}}$), foreground contrast and motion ($\Delta I_{\text{movers}}$) and artifacts ($\Delta I_{\text{artifacts}}$).

The equation does not account for non-standard viewing conditions, or playback frame rates other than the sample frame rate. Artifacts in this version are accounted for in PSNR. $\Delta I_{\text{artifacts}}$ is not used in this version, but reserved for accounting for specific transmission losses such as missing macroblocks in future revisions of this document.

**Ground Sample Distance** ($GSD$) is the geometric mean of the horizontal and vertical components in millimeters at the center of the frame (see Section 7.1).

<table>
<thead>
<tr>
<th>Requirement</th>
<th>The reference benchmark is the instantaneous Video NIIRS shall be 5 when the GSD is 528 millimeters and all other motion image quality factors are high quality as shown in MISB RP 1203.3 Table 1.</th>
</tr>
</thead>
</table>

**Table 1: Reference Video-NIIRS for High Quality Video**

<table>
<thead>
<tr>
<th>Video NIIRS</th>
<th>GSD(mm)</th>
<th>RER</th>
<th>pSNR</th>
<th>camera motion</th>
<th>scene contrast</th>
<th>mover discon. &amp; contrast</th>
<th>artifacts</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>2112</td>
<td>1</td>
<td>40</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>1056</td>
<td>1</td>
<td>40</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>528</td>
<td>1</td>
<td>40</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>6</td>
<td>264</td>
<td>1</td>
<td>40</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>7</td>
<td>132</td>
<td>1</td>
<td>40</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>8</td>
<td>66</td>
<td>1</td>
<td>40</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>9</td>
<td>33</td>
<td>1</td>
<td>40</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>10</td>
<td>17</td>
<td>1</td>
<td>40</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>11</td>
<td>8</td>
<td>1</td>
<td>40</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

**Relative Edge Response** (RER) is a measure of the sharpness of an image (see Section 7.2).
RP 1203.3 Video Interpretability and Quality Measurement and Prediction

<table>
<thead>
<tr>
<th>Requirement</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RP 1203.3-02</td>
<td>When calibration is possible RER shall be estimated using slanted-edge method in accordance with ISO 12233[2].</td>
</tr>
<tr>
<td>RP 1203.3-03</td>
<td>When no calibration is possible RER shall be estimated using the partial-reference technique.</td>
</tr>
<tr>
<td>RP 1203.3-04</td>
<td>When no partial-reference technique is possible RER shall be estimated using the no-reference technique described in MISB RP 1203.</td>
</tr>
</tbody>
</table>

**PSNR** is a measure of the **noise** and **artifacts**. **PSNRc** is the critical point near which digital video rapidly loses interpretability. Video approaching this point is dangerously near the digital cliff. **PSNRc** has been experimentally determined to be 26 dB (see Section 7.3).

<table>
<thead>
<tr>
<th>Requirement</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RP 1203.3-05</td>
<td>For video (motion imagery) with known reference (uncompressed or lightly compressed), PSNR shall be estimated using the known reference.</td>
</tr>
<tr>
<td>RP 1203.3-06</td>
<td>When no reference is available PSNR shall be estimated using the partial-reference technique.</td>
</tr>
<tr>
<td>RP 1203.3-07</td>
<td>When no partial-reference technique is possible, PSNR shall be estimated using the no-reference technique described in MISB RP 1203.</td>
</tr>
</tbody>
</table>

Acquisition and procurement specifiers are strongly advised to use reference based techniques because of their inherent reproducibility and lower variance.

The no-reference techniques are dependent on operator usage and may not necessarily indicate equipment faults. For example, rapid slewing or mode changes cause degradation based on usage not inherent equipment capabilities.

- **ΔI<sub>camera</sub>** (described below) accounts for the negative effects of excessive **camera motion**.
- **ΔI<sub>contrast</sub>** (described below) modifies output based on overall **scene contrast**.
- **ΔI<sub>movers</sub>** (described below) subtracts based on **mover contrast** and **discontinuity**.
- **ΔI<sub>artifacts</sub>** (described below) subtracts based on image **artifacts** such as blockiness due to compression and transmission errors.

Compounding of IR motion imagery to 8-bits is accomplished using the technique in the Appendix of MISB ST 0404 [3].

For Standard Definition (SD) the conversion of RGB color to luminance values is accomplished via NTSC conversion,

\[
L = 0.2989*R + 0.5870*G + 0.1140*B
\]

For Digital High Definition (HD) the conversion of RGB color to luminance values is,

\[
L = 0.2126*R + 0.7152*G + 0.0722*B
\]
When the input luminance values are “studio swing” i.e. restricted to a range of 15-235, they shall be converted to the “full swing” range of 0-255.

All subsequent operations described below are on the luminance values.

A perceptual analysis window is chosen to select the salient features using phase congruency [8]. Phase congruency is not sensitive to variations in image illumination, blurring, and magnification.

The phase congruency image is obtained from the intensity image, “iraw” as

\[ icon = \text{phasecongmono}(iraw) \]

The Matlab function is provided in the Appendix.

Perceptual analysis window specifications as as follows:

<table>
<thead>
<tr>
<th>Requirement</th>
<th>Specification</th>
</tr>
</thead>
<tbody>
<tr>
<td>RP 1203.3-09</td>
<td>The minimum perceptual analysis window size in either the height or width dimension shall be 256.</td>
</tr>
<tr>
<td>RP 1203.3-10</td>
<td>The perceptual analysis window shall have the same width to height ratio as the entire frame.</td>
</tr>
<tr>
<td>RP 1203.3-11</td>
<td>Both the horizontal and vertical dimensions of the perceptual analysis window shall be a multiple of 32.</td>
</tr>
<tr>
<td>RP 1203.3-12</td>
<td>Nine, possibly overlapping, candidate perceptual analysis windows shall be evaluated around the frame center with offsets 20% of overall frame height and center, respectively.</td>
</tr>
<tr>
<td>RP 1203.3-13</td>
<td>Phase congruency shall be calculated for each candidate perceptual analysis window and summed over rows and columns.</td>
</tr>
<tr>
<td>RP 1203.3-14</td>
<td>The perceptual analysis candidate window with the greatest sum shall be selected for additional analysis as described in MISB RP 1203.3.</td>
</tr>
<tr>
<td>RP 1203.3-15</td>
<td>The analysis frame rate, A-FPS, shall be greater than 20% of the sample frame rate.</td>
</tr>
<tr>
<td>RP 1203.3-16</td>
<td>The time between analysis frames shall be less than 1 second.</td>
</tr>
<tr>
<td>RP 1203.3-17</td>
<td>When waves and or similar textures (trees in the wind) are being analyzed, A-FPS shall be set equal to FPS to prevent background motion being interpreted as artifact noise.</td>
</tr>
</tbody>
</table>

Two features are computed from the phase congruency image.

- MICON is the mean of the phase congruency image.
- EICON is the entropy of the phase congruency image.

As an example of A-FPS for 30 FPS video, the analysis frame rate (A-FPS) must be greater than 6 Hz; that is, no less than every 5th frame is to be analyzed.
7.1 **Ground Sample Distance (GSD) calculation**

GSD is calculated from the metadata (e.g. MISB ST 0902 [4] also called MMS). It is important that the GSD-related metadata be accurately re-sampled to be frame synchronous.

A 10% error in GSD causes a 3% error in Video-NIIRS at the benchmark case of 230 mm (Video-NIIRS = 5).

A 10% error in GSD at Video-NIIRS = 8 causes a 2% error in Video-NIIRS; however a 10% GSD error is just 3 millimeters.

Required metadata elements are:

1. \( R = \) Tag #21 in MMS (MISB ST 0902) “Slant Range” in meters
2. \( \theta_h = \) Tag #16 in MMS Field of View Horizontal (degrees)
3. \( \theta_v = \) Tag #17 in MMS Field of View Vertical (degrees)
4. \( \alpha = \) Sensor Principal Axis Elevation Angle Relative to local horizontal plane (radians, computed using procedure outlined in MISB ST 0601[6])

\[
GSD = 2R \sqrt{\frac{\tan(\frac{\pi \theta_h}{360}) \tan(\frac{\pi \theta_v}{360})}{\sin(\alpha)}}
\]

**Equation 2**

7.2 **Relative Edge Response (RER) Instantaneous Measurement**

The cumulative blurring due to atmosphere and image chain can be a primary cause of video degradation. The RER parameter is the second most important after GSD.

7.2.1 **Manual RER when a slanted-edge reference is available**

The ISO 12233[2] blur metric is computed using the method described in [2] and [12].

7.2.2 **Automatic, No-reference RER estimate**

Individual RER estimates are derived from the blind blur metric (BM), the Edge Intensity (EI), the perceptual-RER (pRER), and the Frequency Ratio (FR):

\[
\begin{align*}
\text{RER}(\text{BM}) &= 1.17 - 1.15 \times \text{BM} \\
\text{RER}(\text{EI}) &= -0.28 + 1.3 \times \left( \frac{\text{EI}}{100} \right)^{1/4} \\
\text{RER}(\text{pRER}) &= 0.10 + 0.55 \times \text{pRER} \\
\text{RER}(\text{FR}) &= -0.26 + 3 \times \left( \frac{\text{FR}}{1} \right)^{1/4}
\end{align*}
\]

**Equations 3, 4, 5, 6**

The no-reference RER estimate is the average of the four individual RER estimates,
RER = (RER(BM) + RER(EI) + RER(pRER) + RER(FR))/4 \textbf{Equation 7}

The combined estimate has a standard deviation of error of 0.1 for normal scene content with sufficient spatial energy. An RER error of 0.1 equates to 0.25 Video-NIIRS error.

A special intermittent error condition can occur when the scene does not contain any spatial energy such as sky, smooth water, or featureless desert. In this case, the instantaneous RER estimate may be erroneously low; however, this temporary condition will be detected and filtered out in subsequent processing. See Section 8.1 “Filtering Instantaneous Estimates”.

Sustained operation over sea state 0 or 1 requires additional research. Sea states greater than 2 should generate enough features for RER computation, although this remains to be tested.

Individual RER estimates are described below.

7.2.2.1 Blind Blur Metric, BM

The blur metric is estimated using the method described in [9].

7.2.2.2 Edge Intensity, EI

Edge intensity is,

$EI = \frac{1}{RC} \sum_{i,j} \sqrt{g(i,j)^2 + g(i,j)^2}$ \textbf{Equation 8}

where, $g_x$ and $g_y$ are the Sobel horizontal and vertical filtered versions of the input image.

7.2.2.3 Perceptual-RER (pRER)

Perceptual-RER is a blind technique that uses phase congruency and morphological operations to estimate the edge response. The matlab code is listed in the Appendix.

7.2.2.4 Frequency Ratio, Fr

The Frequency Ratio is computed by taking the ratio of high-pass to low-pass energy.

<table>
<thead>
<tr>
<th>Requirement</th>
</tr>
</thead>
<tbody>
<tr>
<td>RP 1203.3-18</td>
</tr>
</tbody>
</table>

Matlab code listing is in the Appendix.

7.2.2.5 Partial-Reference RER Metric

At the source each frame is divided into regions. Each region is analyzed for edge energy. Once a qualified, high-edge energy region is found, the blur is estimated. The location of the region and the blur value is sent.

At the receiver a similar computation is performed. Comparison of the blur estimates indicates distortion in the channel.
7.3 Noise Measurement, PSNR

7.3.1 Noise Metric when a reference is available

Standard PSNR is used when a reference is available.

PSNR is defined via the mean squared error (MSE), which for two \( m \times n \) monochrome images \( I \) and \( K \) where one of the images is considered a noisy approximation of the other is defined as:

\[
MSE = \frac{1}{mn} \sum_{i=0}^{m-1} \sum_{j=0}^{n-1} [(i, j) - K(i, j)]^2
\]

Equation 9

The PSNR is defined as:

\[
PSNR = 10 \log_{10}(MAX_i/MSE)
\]

Equation 10

\( MAX_i \) is the maximum possible pixel value of the image. When the pixels are represented using 8 bits per sample, this is 255.

7.3.2 No-Reference Instantaneous PSNR Estimate

The PSNR, when no reference is available, is estimated by applying a weighted sum to a feature vector and its pairwise components.

Additional accuracy can be gained by averaging over multiple frames. See Section 8.1 “Filtering Instantaneous Estimates”.

The feature vector has the following 11-elements. The 3rd element, “Evar” is described in Reference [5] which also points to source code.

\[
\begin{align*}
V(1) &= Fr = frequency\_ratio(iraw, 256, fc); \\
V(2) &= BM = blurMetric(iraw); % blur metric, \\
V(3) &= Evar = evar(iraw); \\
V(4) &= MICON = mean(mean(icon)); \\
V(5) &= EICON = entropy(icon); \\
V(6) &= EI = edge\_intensity(iraw); \\
V(7) &= STD = std(double(iraw())); & \text{standard deviation of intensity} \\
V(8) &= MSSIM = SSIM \text{ using stabilization} \\
V(9) &= LAMD = lamd; \\
V(10) &= BLOCKV = r; \\
V(11) &= GM = sum(abs(TCUM));
\end{align*}
\]

The number of coefficients is \( n(n-1)/2 + n + 1 \). For \( n=11 \) there are 67 coefficients. The coefficients are controlled separately from this document.

The PSNR coefficients shown in Table 3 in the Appendix are from the file “psnrcofs001.csv”.

<table>
<thead>
<tr>
<th>Requirement</th>
<th>The version number of the PSNR coefficient file shall be indicated in the metadata using the “PSNR Coefficient Identifier” in accordance with MISB ST 1108[5].</th>
</tr>
</thead>
<tbody>
<tr>
<td>RP 1203.3-19</td>
<td></td>
</tr>
</tbody>
</table>

27 February 2014 Motion Imagery Standards Board
PSNR coefficients are controlled separately from this document. Multiple PSNR coefficient files are needed as the motion imagery training set encompasses more situations. For example, it may be that each IR band has its own PSNR coefficient file.

### 7.3.3 Partial-Reference Noise Metric

At the source each frame is divided into regions. Each region is analyzed for edge energy. Once a qualified, homogenous, low-edge energy region is found the noise is estimated. The location of the region and the noise value is inserted into the stream metadata and transmitted along with the stream.

At the receiver a similar computation is performed. Comparison of the noise estimates indicates distortion in the channel.

### 7.4 Camera MOTION and Scene Discontinuity $\Delta I_{\text{camera}}$

Excessive translation between analysis frames causes a loss of interpretability. Scene translation discontinuity is defined as,

$$\Delta_{\text{scene-}\text{xy}}=(A-FPS/FPS)(1/Fs),$$

where Fs is the fractional overlap between analysis frames. For example, if there is 50\% overlap between the current frame and the previous analysis frame, then Fs=0.5 and 1/Fs=2. If the analysis frames are 5 video frames apart then (A-FPS/FPS)=(1/5). For this example, $\Delta_{\text{scene-xy}}=2/5=0.4$.

Computation of scene overlap between analysis frames is not needed when the metadata indicates rapid slewing. Below is a simple test for Fs=0 based on the metadata.

If Fs=0, $\Delta_{\text{scene-xy}}$ is set to “realmax” (largest real number representable by the system).

Fast algorithm assumes 4-corner metadata is stored in r1 vector for current frame and r2 vector for previous frame, e.g. for current frame:

- $r1(1,1)=\text{lon}1$
- $r1(2,1)=\text{lon}2$
- $r1(3,1)=\text{lon}3$
- $r1(4,1)=\text{lon}4$
- $r1(1,2)=\text{lat}1$
- $r1(2,2)=\text{lat}2$
- $r1(3,2)=\text{lat}3$
- $r1(4,2)=\text{lat}4$

then perform the following tests:

```matlab
maxX1 = max(r1(:, 1));
maxY1 = max(r1(:, 2));
minX1 = min(r1(:, 1));
minY1 = min(r1(:, 2));
maxX2 = max(r2(:, 1));
```
maxY2 = max(r2(:, 2));
minX2 = min(r2(:, 1));
minY2 = min(r2(:, 2));

boundingBox=0; %initialize to assume no-overlap
if(minX1 <= maxX2) %Do a fast bounding box intersection to quickly rule out no intersection
    if(maxX1 >= minX2)
        if(maxY1 >= minY2)
            if(minY1 <= maxY2)
                boundingBox = 1;
            end
        end
    end
end

---

<table>
<thead>
<tr>
<th>Requirement</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>RP 1203.3-20</td>
<td>If the quick metadata test indicates there are zero common pixels between analysis frames (Fs=0 because boundingBox=0), and the ground speed of the center point is greater than 322 Km/H (200 MPH), then the instantaneous Video-NIIRS and the instantaneous Video Quality shall be set to zero during the slewing interval. REASON CODE: SLEW</td>
</tr>
</tbody>
</table>

Otherwise, if the metadata test indicates overlap, then global camera motion is measured between the current analysis window (iraw) and the previous analysis window (iraw_old) using a rigid model of rotation and translation implemented by the matlab function, “opticalflow” shown in the Appendix.

The following pertains to “opticalflow.m”:

The 3rd argument controls the Region-of-Interest which is set to the entire size of the analysis window. The last argument controls a downsampling parameter which is not used. The called functions are also shown below. Downsampling is an integer number. More motion between analysis frames requires a larger downsampling number. Translational motion of up to 25 pixels between analysis frames can be estimated with downsampling parameter set to 5.

The output arguments of interest are the vertical and horizontal translations shifts in the “Tcum” vector, and the variable “iga” which is the previous analysis window rotated and translated to match the current one. “Acum” is not used.

A rolling buffer of Tcum(1) is stored in “vvec” and analyzed to determine video high frequency instability in the vertical dimension.

A rolling buffer of Tcum(2) is stored in “hvec” and analyzed to determine video high frequency instability in the horizontal dimension.

The length of vvec and hvec rolling buffers shall be 30.
A rolling buffer stores a new measurement at its head and deletes the oldest measurement at its tail as new data is available to be stored.

The standard deviation of the detrended rolling buffer is the video spatial jitter metric.

Detrending is a linear fit that most nearly makes the data zero-mean.

<table>
<thead>
<tr>
<th>Requirement</th>
<th>If either the vertical or horizontal spatial jitter metric is greater than 16, then the instantaneous Video-NIIRS and the instantaneous Video Quality shall be set to zero. REASON CODE: JITTER</th>
</tr>
</thead>
</table>

### 7.5 Overall Scene Contrast and Dynamic Range, $\Delta I_{contrast}$

Overall scene contrast is computed by taking the histogram of the analysis frame.

<table>
<thead>
<tr>
<th>Requirement</th>
<th>If the 80% percentile point of the overall scene contrast histogram is above the intensity value of 220, then the instantaneous Video-NIIRS and the instantaneous Video Quality shall be set to zero. REASON CODE: OVERSAT.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>If the width of the histogram is too narrow as measured by the 2nd moment then the instantaneous Video-NIIRS and the instantaneous Video Quality shall be set to zero. REASON CODE: DYNAMIC RANGE</td>
</tr>
</tbody>
</table>

As an example of REQ. RP 1203.3-24, if $\sigma_{\text{intensity}} < 15$ (8-bit imagery) then instantaneous Video-NIIRS and the instantaneous Video Quality will be set to zero. Given values are for 8-bit imagery. Appropriate adjustments need to be made for other bit depths.

### 7.6 Mover Contrast and Discontinuity $\Delta I_{\text{movers}}$

Methods for detection and tracking of movers are outside the scope of this document. However, if information concerning movers is available (e.g. Video Moving Target Indicator, MISB ST 0903 [15]) can be used to more accurately gauge the interpretability. If this information, or equivalent, is not available, $\Delta I_{\text{movers}} = 0$.

Mover contrast and discontinuity is grouped together because the detection limit depends on size, speed, and contrast.

The mean modulation contrast surrounding the mover is computed in at least 4 directions,

$$ C_m = \frac{|{\overline{L_{out}}} - {\overline{L_{in}}}|}{{\overline{L_{out}}} + {\overline{L_{in}}}} \quad \text{Equation 12} $$

where,

$$ {\overline{L_{out}}} = \frac{1}{K} \sum_{k=1}^{K} L_{out}(k), \quad \text{and} \quad {\overline{L_{in}}} = \frac{1}{K} \sum_{k=1}^{K} L_{in}(k) \quad \text{Equation 13} $$
$C_m$ is the mover contrast, $L_{out}$ and $L_{in}$ are the gamma-corrected pixel luminance values inside and outside the mover contour.

The threshold contrast is predicted by the following equation (R-squared = 0.98)

$$C_t = \text{abs}(0.184/(\sqrt{x_0} + 0.8817*x_1/(363.54 + (158.91 + x_1)/x_0)) - 0.02383)$$

Equation 14

Where $x_0$ is the target size in pixels and $x_1$ is the target speed in pixels per-second.

The formula is only valid for standard viewing conditions, where the imagery is displayed at its native resolution on a device from a distance that does not degrade the ability to perceive the inherent detail. More precise definitions for standard viewing conditions are available from the NGA Image Quality and Utility (NIQU) organization.

If the measured contrast is less than the predicted threshold contrast (based on the size and speed of the detected object) then the Video Interpretability is reduced according to the following.

Define $C_r = \log_{10}(C/C_t)$ as the log of the ratio of the detected mover contrast to threshold.

Let N be number of detected movers with $C_r$ less than 0.5.

$\delta I$ = amount to add to Video NIIRS estimate

**Interpretability Advanced Pooling**

If all detected Movers have $C_r$ greater than 0.5 then do not modify interpretability.

If all detected Movers have $C_r$ less than 0 reduce I by 1.0.

Otherwise $\delta I = -N*0.2$

For example:

If one detected mover has $C_r$ less than 0.5 $\delta I = -0.2$

If two detected movers have $C_r$ less than 0.5 $\delta I = -0.4$

If three detected movers have $C_r$ less than 0.5 $\delta I = -0.6$

If four detected movers have $C_r$ less than 0.5 $\delta I = -0.8$

In no case can Interpretability be less than 0.

### 7.7 Artifact Metrics

#### 7.7.1 Blockiness Detector

**BLOCKV** is feature 10 in PSNR feature vector (Section 7.3.2). Code listing is in the Appendix.
7.7.2 Modified SSIM (M-SSIM)

The structural similarity (SSIM) index is a method for measuring the similarity between two images. It is usually applied between an uncompressed reference and the image being tested. However, the M-SSIM metric applies the SSIM between the current analysis window and the previous analysis window that has been warped by “opticalflow.m” to match. This technique is not as sensitive as SSIM when used with a reference, but it is capable of detecting transitory artifacts.

M-SSIM is feature 08 in the PSNR feature vector (Section 7.3.2).

If Video Moving Target Indicator (VMTI), MISB ST 903 [15] amplifying data is available, movers are segmented and removed from the M-SSIM calculation.

A-FPS shall be set equal to FPS when video of waves and or similar textures are being analyzed to prevent background motion being interpreted as artifact noise (see REQ. RP 1203.3-17).

<table>
<thead>
<tr>
<th>Requirement</th>
</tr>
</thead>
<tbody>
<tr>
<td>RP 1203.3-25</td>
</tr>
</tbody>
</table>

Algorithm 1:

```
[mr mc]=size(iga); %previous analysis frame warped to match current
igar=iga(32:mr-32,32:mc-32);
igr=iraw(32:mr-32,32:mc-32); % iraw is current analysis frame
```

The modified SSIM is computed by downsampling using Matlab imresize function and then calling the ssim function.

```
M-SSIM=ssim(imresize(igar,.5),imresize(igr,.5)); see Reference [16] for ssim.m
```

WARNING: Burned in metadata can make M-SSIM readings too low. Simple metadata removal is not adequate for metadata that “floats”. That is it neither is tied to the ground, or the camera. Cross-hairs float.

7.7.3 LAMBDA

This artifact metric is computed by warping the previous analysis frame to the current analysis frame. Once the two frames are aligned the PSNR is computed between them using the current analysis frame as the reference.

<table>
<thead>
<tr>
<th>Requirement</th>
</tr>
</thead>
<tbody>
<tr>
<td>RP 1203.3-26</td>
</tr>
</tbody>
</table>

LAMBDA is feature 09 in PSNR feature vector.

Additional filtering is provided to reduce the effect of floating metadata. The filtering ensures only a strong artifact registers. Otherwise, LAMBDA is set to zero.

```
dim=(abs(double(igr)-igar)); %current and warped frame difference
```
am=sum(sum(dim))/(m*n);  
if km==1  %initialize for first frame
   ams(1)=am;
   amd=0;
end;
alpha=.35;  %filtering parameter
if km>1
   ams(km)=alpha*am+(1-alpha)*ams(km-1);  %exponential average
   amd=am-ams(km);  %instantaneous relative to average
end;
if amd<0.01
   amd=0.01;
end;
lambda=log10(amd);
if lambda<0.1  %hard threshold prevents false artifact alarms
   lambda=0;
end;
LAMBDA=lambda;

If Video Moving Target Indicator (VMTI), MISB ST 0903 [15] amplifying data is available, movers are segmented and removed from the LAMBDA calculation.

If VMTI information is not available, some other method must be used to detect and remove mover energy from the LAMBDA estimate.

A-FPS shall be set equal to FPS when video of waves and or similar textures are being analyzed to prevent background motion being interpreted as artifact noise (see REQ. RP 1203.3-17).

7.7.4 Instantaneous Global Motion (GM)  

\[ GM = \text{sum}(|T_{cum}|) \]

\( GM \) is feature 11 in PSNR feature vector.  

opticalflow.m listing is in Appendix.

The metrics described above are used to predict PSNR and are not included in \( \Delta I_{\text{artifacts}} \).  

\( \Delta I_{\text{artifacts}} \) is reserved for future use to describe the effects of packet loss and inclusion of additional metadata such as encoder psnr, and decoder error message.

\( \Delta I_{\text{artifacts}} \) shall be set to zero in this version.

8 Normative Instantaneous Video Quality

The probability of each quality class (Excellent, Good, Fair, Poor, Bad) is computed using the following logistic model.
The video quality is assigned one of five values corresponding to BAD, POOR, FAIR, GOOD, or EXCELLENT. The probability of each value is estimated by calculating a feature vector and applying appropriate coefficients as shown in the following equations:

\[
\Pr(Y_i = 1 \text{ (BAD)}) = \frac{\exp(\beta_{1}^{'} \cdot X_i)}{1 + \sum_{k=1}^{K-1} \exp(\beta_{k}^{'} \cdot X_i)} \quad \text{Equation 15}
\]

\[
\Pr(Y_i = 2 \text{ (POOR)}) = \frac{\exp(\beta_{2}^{'} \cdot X_i)}{1 + \sum_{k=1}^{K-1} \exp(\beta_{k}^{'} \cdot X_i)} \quad \text{Equation 16}
\]

\[
\Pr(Y_i = 3 \text{ (FAIR)}) = \frac{\exp(\beta_{3}^{'} \cdot X_i)}{1 + \sum_{k=1}^{K-1} \exp(\beta_{k}^{'} \cdot X_i)} \quad \text{Equation 17}
\]

\[
\Pr(Y_i = 4 \text{ (GOOD)}) = \frac{\exp(\beta_{4}^{'} \cdot X_i)}{1 + \sum_{k=1}^{K-1} \exp(\beta_{k}^{'} \cdot X_i)} \quad \text{Equation 18}
\]

\[
\Pr(Y_i = 5 \text{ (EXCELLENT)}) = \frac{1}{1 + \sum_{k=1}^{K-1} \exp(\beta_{k}^{'} \cdot X_i)} \quad \text{Equation 19}
\]

\(X_i\) is a feature vector for the ith frame. \(Y_i\) is the estimated quality class for the ith frame.

The class with the greatest probability is selected if it exceeds a minimum threshold probability. The minimum threshold probability is 0.70.

For non-reference (blind) modes:

If instantaneous Video Quality does not meet threshold probability, then a negative number is also returned for Video Interpretability.

If instantaneous Video Quality is “BAD” then instantaneous Video Interpretability is set to zero. REASON CODE: BAD

The probability for each class is computed using the function multinomialLogisticProbability.m (in the Appendix).

The quality coefficients shown are from the file “qualcofs003.csv”. The version number of the coefficient file shall be indicated in the metadata using the “Quality Coefficient Identifier” (see ST 1108). Quality coefficients are controlled separately from this document.

Multiple Quality coefficient files are needed as the training motion imagery set increases. For example, it may be that each IR band has its own Quality coefficient file.

Conversion of the 5-point scale to a 100-point scale is accomplished by multiplying by 20.

Conversion of 100-point Quality values to 5-point scale is accomplished by dividing by 20 and rounding.
Table 2: Video Quality Multinomial Logistic Likert Coefficients

<table>
<thead>
<tr>
<th>No.</th>
<th>Multinomial Logistic Coefficients (*&quot;beta&quot; in probability equation)</th>
<th>Feature Description</th>
<th>Feature Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>2.2372 -3.5973 -4.7296 1.0965</td>
<td>constant term</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>65.931 95.154 50.395 -120.82</td>
<td>frequency_ratio(iraw,256,fc);</td>
<td>RER section</td>
</tr>
<tr>
<td>2</td>
<td>8.865 -10.885 -12.388 10.187</td>
<td>blurMetric(iraw);%blur metric</td>
<td>RER section</td>
</tr>
<tr>
<td>3</td>
<td>0.39482 0.38715 0.039475 0.89175</td>
<td>evar(iraw);</td>
<td>PSNR section</td>
</tr>
<tr>
<td>4</td>
<td>-1408 -744.72 -193.23 -406.03</td>
<td>mean(mean(icon));</td>
<td>phase congruence Ref [7]</td>
</tr>
<tr>
<td>5</td>
<td>23.268 19.658 9.5779 4.516</td>
<td>entropy(icon);</td>
<td>perceptual window section</td>
</tr>
<tr>
<td>6</td>
<td>0.075755 0.23649 0.2627 -0.0731</td>
<td>edge_intensity(iraw);</td>
<td>see RER section</td>
</tr>
<tr>
<td>7</td>
<td>-0.27688 -0.19997 -0.19399 0.09465</td>
<td>std(double(iraw(:)));</td>
<td>std. of intensity</td>
</tr>
<tr>
<td>8</td>
<td>-25.824 -21.343 -5.6556 2.106</td>
<td>ssim(imresize(igar,.5),imresize(igar,.5));</td>
<td>Reference [15] and Artifacts Section</td>
</tr>
<tr>
<td>9</td>
<td>-7.7294 -6.7376 0.40658 7.5199</td>
<td>lamd;</td>
<td>Artifacts Section</td>
</tr>
<tr>
<td>10</td>
<td>0.66441 0.42854 0.39753 0.2418</td>
<td>blockiness;</td>
<td>Artifacts Section</td>
</tr>
<tr>
<td>11</td>
<td>-3.6211 -12.957 -14.232 -4.2786</td>
<td>perceptual_RER4(iraw);</td>
<td>RER section</td>
</tr>
<tr>
<td>12</td>
<td>-0.08494 -0.0607 -0.03243 -0.0031</td>
<td>sum(abs(Tcum));</td>
<td>Motion section</td>
</tr>
</tbody>
</table>

8.1 Filtering Instantaneous Estimates

Averaging is one type of filtering operation. Averaging provides a more stable estimate of the interpretability and quality that is representative over a duration.

Trimmed mean is applied to the instantaneous values to remove outliers in the averaging buffer. The trimmed mean parameter alpha=0.05 discards spikes.

Averages are not valid across scene change boundaries. Scene change is in progress whenever the 4-corners between adjacent analysis frames do not overlap, or when the metadata indicates FOV change.

Averages are not valid across mode changes e.g. IR-to-EO or EO-to-IR.

Averages are not valid when switching from/to Black-Hot from/to White-Hot.

Whenever any of these changes are detected, the current averaging duration ends and the Video Interpretability, Quality, and Rating Duration are inserted into the metadata stream per ST 1108 [5]. Interpretability and Quality Method ID shall be set to 0.

The Video Interpretability equation is for EO only. Use of the EO equation for IR modes is only to give a sense of relative interpretability changes.

Statistical Quality Control shall be established on the instantaneous video quality estimates to enable alerts based on abnormal readings outside of expected normal variations.

Exceedances are defined as any instantaneous measurement that is zero, or statistically out of normal bounds.
If the measurement is zero, an exceedance count is incremented based on REASON CODE. Number of exceedances per-hour, per-minute, and per-second are kept.

<table>
<thead>
<tr>
<th>Requirement</th>
<th>Instantaneous Video Quality shall be monitored to get number of “BAD” frames per-hour, per-minute, and per-second.</th>
</tr>
</thead>
<tbody>
<tr>
<td>RP 1203.3-27</td>
<td></td>
</tr>
<tr>
<td>RP 1203.3-28</td>
<td>Instantaneous Video Quality shall be monitored to get number of “POOR” frames per-hour, per-minute, and per-second.</td>
</tr>
<tr>
<td>RP 1203.3-29</td>
<td>Instantaneous Video Quality shall be monitored to get number of “FAIR” frames per-hour, per-minute, and per-second.</td>
</tr>
<tr>
<td>RP 1203.3-30</td>
<td>Instantaneous Video Quality shall be monitored to get number of “GOOD” frames per-hour, per-minute, and per-second.</td>
</tr>
<tr>
<td>RP 1203.3-31</td>
<td>Instantaneous Video Quality shall be monitored to get number of “EXCELLENT” frames per-hour, per-minute, and per-second.</td>
</tr>
<tr>
<td>RP 1203.3-32</td>
<td>Instantaneous Video Quality shall be monitored to get number of “NOT-RATED” frames per-hour, per-minute, and per-second.</td>
</tr>
</tbody>
</table>

9 Informative: Video IQ Verification Procedure

Validation versus Verification
Validation assesses that the correct metric is selected to measure the performance of the task in question. Verification assures that the selected metric operates correctly.

Video-NIIRS and Quality are metrics that predict interpretability and quality as rated by humans. Using Video-NIIRS and Quality as metrics for other tasks could be sub-optimal, or invalid.

Verification that the values of the IQ keys actually represent the interpretability and quality of the motion imagery is accomplished as follows.

Assemble Test Set
Test set shall be multiple source video segments with varying content. Each source segment shall be at least 10-seconds and not contain extreme zoom changes. Segments shall differ in content with respect to Ground Sample Distance (GSD) and quality. Additional segments can be derived from the source segments to simulate distortion or processing improvements.

Perform Video Interpretability and Quality Ratings
Segments are viewed and rated to obtain Video-NIIRS and DMOS ratings. Statistical power, resolution, accuracy and precision are increased with additional raters. Suggest at least five raters for minor studies and at least 15 for more important ones.

If comparing source segments to derived segments, both shall be presented to the rater.

Scaling Quality Subjective Data
Quality score scaling shall be performed for each subject individually across all data points.

A scaled rating is calculated as follows: scaled rating = (raw difference score - Min) / (Max - Min ) where Max = largest raw difference score for that subject and Min = minimum raw
difference score for that subject. Note that the Max difference corresponds to the poorest judged quality, and Min corresponds to the best judged quality.

**Eliminating subjects**
Data from raters that have clearly misinterpreted the directions and are rating using opposite polarity can be eliminated.

**Objective Data Analysis**
Performance of the objective models is evaluated with respect to three aspects:

- **prediction accuracy** – the ability to predict the subjective quality ratings with low error,
- **prediction monotonicity** – the degree to which the model’s predictions agree with the relative magnitudes of subjective quality ratings and
- **prediction consistency** – the degree to which the model maintains prediction accuracy over the range of video test sequences, i.e., that its response is robust with respect to a variety of video impairments.

**Metrics relating to Prediction Accuracy, Metric 1:** The Pearson linear correlation coefficient between predicted and manual. Pearson values greater than 0.8 indicate good accuracy.

**Metrics relating to Prediction Monotonicity, Metric 2:** Spearman rank order correlation coefficient between predicted and manual. Spearman rank order correlation coefficients indicate good monotonicity.

**Metrics relating to Prediction Consistency, Metric 3:** Outlier Ratio of “outlier-points” to total points $N$.

\[
\text{Outlier Ratio} = \frac{\text{total number of outliers}}{N} \quad \text{Equation 20}
\]

where an outlier is a point for which: $\text{ABS}[\text{Error}[i]] > 2 \times \text{StandardError}[i]$.

Twice the Standard Error is used as the threshold for defining an outlier point. Outlier ratio less than 10% indicates good prediction consistency.
10 Appendix: Matlab Code

10.1 phasecongmono.m

% PHASECONGMONO - phase congruency of an image using monogenic filters
% This code is considerably faster than PHASECONG3 but you may prefer the
% output from PHASECONG3's oriented filters.
% There are potentially many arguments, here is the full usage:
% [PC or ft T] = ...
% phasecongmono(im, nscale, minWaveLength, mult, ...
% sigmaOnf, k, cutOff, g, noiseMethod)
% However, apart from the image, all parameters have defaults and the
% usage can be as simple as:
% phaseCong = phasecongmono(im);
% Arguments:
% Default values  Description
% nscale           5  - Number of wavelet scales, try values 3-6
% minWaveLength    3  - Wavelength of smallest scale filter.
% mult             2.1 - Scaling factor between successive filters.
% sigmaOnf         0.55 - Ratio of the standard deviation of the Gaussian
% describing the log Gabor filter's transfer
% function in the frequency domain to the filter
% center frequency.
% k                2.0 - No of standard deviations of the noise energy
% beyond the mean at which we set the noise
% threshold point.
% You may want to vary this up to a value of 10 or 20 for noisy images
% noiseMethod  -1  - Parameter specifies method used to determine
% noise statistics.
% -1 use median of smallest scale filter
% responses
% -2 use mode of smallest scale filter responses
% 0+ use noiseMethod value as the fixed noise
% threshold
% A value of 0 will turn off all noise
% compensation.
% cutOff           0.5 - The fractional measure of frequency spread
% below which phase congruency values get
% penalized.
% g                10 - Controls the sharpness of the transition in
% the sigmoid function used to weight phase
% congruency for frequency spread.
% Returned values:
% PC  - Phase congruency indicating edge significance
% or  - Orientation image in integer degrees 0-180,
positive anticlockwise.
0 corresponds to a vertical edge, 90 is horizontal.

ft - Local weighted mean phase angle at every point in the
image. A value of pi/2 corresponds to a bright line, 0
corresponds to a step and -pi/2 is a dark line.

T - Calculated noise threshold (can be useful for
diagnosing noise characteristics of images). Once you know
this you can then specify fixed thresholds and save some
computation time.

Notes on specifying parameters:
The parameters can be specified as a full list eg.
>> PC = phasecongmono(im, 5, 3, 2.5, 0.55, 2.0);

or as a partial list with unspecified parameters taking on default values
>> PC = phasecongmono(im, 5, 3);

or as a partial list of parameters followed by some parameters specified
via a
keyword-value pair, remaining parameters are set to defaults, for example:
>> PC = phasecongmono(im, 5, 3, 'k', 2.5);

The convolutions are done via the FFT. Many of the parameters relate to
the
specification of the filters in the frequency plane. The values do not
seem
to be very critical and the defaults are usually fine. You may want to
experiment with the values of 'nscales' and 'k', the noise compensation
factor.

Notes on filter settings to obtain even coverage of the spectrum

sigmaOnf  .85  mult 1.3
sigmaOnf  .75  mult 1.6  (filter bandwidth ~1 octave)
sigmaOnf  .65  mult 2.1
sigmaOnf  .55  mult 3  (filter bandwidth ~2 octaves)

See Also:  PHASECONG, PHASECONG2, PHASECONG3, PHASESYMMONO, GABORCONVOLVE,
PLOTGABORFILTERS

References:

Peter Kovesi, "Image Features From Phase Congruency". Videre: A
Journal of Computer Vision Research. MIT Press. Volume 1, Number 3,

Michael Felsberg and Gerald Sommer, "A New Extension of Linear Signal
Processing for Estimating Local Properties and Detecting Features". DAGM
Symposium 2000, Kiel

Michael Felsberg and Gerald Sommer. "The Monogenic Signal" IEEE
Transactions on Signal Processing, 49(12):3136-3144, December 2001

Peter Kovesi, "Phase Congruency Detects Corners and Edges". Proceedings
% DICTA 2003, Sydney Dec 10-12

% August 2008    Initial version developed from phasesymmono and phasecong2
% where local phase information is calculated via Monogenic
% filters. Simplification of noise compensation to speedup
% execution. Options to calculate noise statistics via median
% or mode of smallest filter response.
% April 2009      Return of T for 'instrumentation' of noise
% detection/compensation. Option to use a fixed threshold.
% June 2009       20% Speed improvement through calculating phase deviation
% via
%                acos() rather than computing cos(theta)-|sin(theta)| via dot
%                and cross products. Also much smaller memory footprint

% Copyright (c) 1996-2009 Peter Kovesi
% School of Computer Science & Software Engineering
% The University of Western Australia
% pk @ csse uwa edu au
% http://www.csse.uwa.edu.au/
% Permission is hereby granted, free of charge, to any person obtaining a
% copy
% of this software and associated documentation files (the "Software"), to deal
% in the Software without restriction, subject to the following conditions:
% The above copyright notice and this permission notice shall be included in all
% copies or substantial portions of the Software.
% The software is provided "as is", without warranty of any kind.

function [PC, or, ft, T] = phasecongmono(varargin)

% Get arguments and/or default values
[im, nscale, minWaveLength, mult, sigmaOnf, k, ...
 noiseMethod, cutOff, g] = checkargs(varargin(:));

epsilon = .0001; % Used to prevent division by zero.
[rows,cols] = size(im);
IM = fft2(im); % Fourier transform of image

sumAn = zeros(rows,cols); % Matrix for accumulating filter response
sumf = zeros(rows,cols);
sumh1 = zeros(rows,cols);
sumh2 = zeros(rows,cols);

% Pre-compute some stuff to speed up filter construction

% Set up u1 and u2 matrices with ranges normalised to +/- 0.5
% The following code adjusts things appropriately for odd and even values
% of rows and columns.
if mod(cols,2)
    xrange = [-(cols-1)/2:(cols-1)/2]/(cols-1);
else
    xrange = [-cols/2:(cols/2-1)]/cols;
end
if mod(rows,2)
    yrange = [-(rows-1)/2:(rows-1)/2]/(rows-1);
else
    yrange = [-rows/2:(rows/2-1)]/rows;
end
[u1,u2] = meshgrid(xrange, yrange);

u1 = ifftshift(u1);   % Quadrant shift to put 0 frequency at the corners
u2 = ifftshift(u2);

% Compute frequency values as a radius from centre (but quadrant shifted)
radius = sqrt(u1.^2 + u2.^2);

% Get rid of the 0 radius value in the middle (at top left corner after
% fftshifting) so that taking the log of the radius, or dividing by the
% radius, will not cause trouble.
radius(1,1) = 1;

% Construct the monogenic filters in the frequency domain. The two
% filters would normally be constructed as follows
%   H1 = i*u1./radius;
%   H2 = i*u2./radius;
% However the two filters can be packed together as a complex valued
% matrix, one in the real part and one in the imaginary part. Do this by
% multiplying H2 by i and then adding it to H1 (note the subtraction
% because i*i = -1). When the convolution is performed via the fft the
% real part of the result will correspond to the convolution with H1 and
% the imaginary part with H2. This allows the two convolutions to be
% done as one in the frequency domain, saving time and memory.
H = (1i*u1 - u2)./radius;

% The two monogenic filters H1 and H2 are not selective in terms of the
% magnitudes of the frequencies. The code below generates bandpass
% log-Gabor filters which are point-wise multiplied by IM to produce
% different bandpass versions of the image before being convolved with H1
% and H2

% First construct a low-pass filter that is as large as possible, yet
% falls
% away to zero at the boundaries. All filters are multiplied by
% this to ensure no extra frequencies at the 'corners' of the FFT are
% incorporated as this can upset the normalisation process when
% calculating phase congruency
lp = lowpassfilter([rows,cols],.4,10);    % Radius .4, 'sharpness' 10
for s = 1:nscale
wavelength = minWaveLength*mult^(s-1);
fo = 1.0/wavelength;                  % Centre frequency of filter.
logGabor = exp((-log(radius/fo)).^2) / (2 * log(sigmaOnf)^2));
logGabor = logGabor.*lp;              % Apply low-pass filter
logGabor(1,1) = 0;                    % Set the value at the 0
frequency point of the filter          % back to zero (undo the radius fudge).

IMF = IM.*logGabor;                   % Bandpassed image in the frequency domain.
f = real(ifft2(IMF));                 % Bandpassed image in spatial domain.

h = ifft2(IMF.*H);                    % Bandpassed monogenic filtering, real part
of h contains                         % convolution result with h1, imaginary
part                                    % contains convolution result with h2.

h1 = real(h);
h2 = imag(h);
An = sqrt(f.^2 + h1.^2 + h2.^2);     % Amplitude of this scale component.
sumAn = sumAn + An;                  % Sum of component amplitudes over
scale.
sumf  = sumf  + f;
sumh1 = sumh1 + h1;
sumh2 = sumh2 + h2;

% At the smallest scale estimate noise characteristics from the
% distribution of the filter amplitude responses stored in sumAn.
% tau is the Rayleigh parameter that is used to describe the
% distribution.
if s == 1
  if noiseMethod == -1     % Use median to estimate noise
    tau = median(sumAn(:))/sqrt(log(4));
  elseif noiseMethod == -2 % Use mode to estimate noise statistics
    tau = rayleighmode(sumAn(:));
  end
  maxAn = An;
else
  % Record maximum amplitude of components across scales. This is
  % needed % to determine the frequency spread weighting.
  maxAn = max(maxAn,An);
end

end   % For each scale

% Form weighting that penalizes frequency distributions that are
% particularly narrow. Calculate fractional 'width' of the frequencies
% present by taking the sum of the filter response amplitudes and
% dividing % by the maximum component amplitude at each point on the image. If
% there is only one non-zero component width takes on a value of 0, if
% all components are equal width is 1.
width = (sumAn./(maxAn + epsilon) - 1) / (nscale-1);
% Now calculate the sigmoidal weighting function.
weight = 1.0 ./ (1 + exp((cutOff - width)*g));

% Automatically determine noise threshold
% Assuming the noise is Gaussian the response of the filters to noise will
% form Rayleigh distribution. We use the filter responses at the
% smallest
% scale as a guide to the underlying noise level because the smallest
% scale
% filters spend most of their time responding to noise, and only
% occasionally responding to features. Either the median, or the mode, of
% the distribution of filter responses can be used as a robust statistic to
% estimate the distribution mean and standard deviation as these are
% related
% to the median or mode by fixed constants. The response of the larger
% scale filters to noise can then be estimated from the smallest scale
% filter response according to their relative bandwidths.
% This code assumes that the expected reponse to noise on the phase
% congruency calculation is simply the sum of the expected noise
% responses
% of each of the filters. This is a simplistic overestimate, however
% these
% two quantities should be related by some constant that will depend on
% the
% filter bank being used. Appropriate tuning of the parameter 'k' will
% allow you to produce the desired output. (though the value of k seems
to
% be not at all critical)

if noiseMethod >= 0     % We are using a fixed noise threshold
    T = noiseMethod;    % use supplied noiseMethod value as the threshold
else
    % Estimate the effect of noise on the sum of the filter responses as
    % the sum of estimated individual responses (this is a simplistic
    % overestimate). As the estimated noise response at successive scales
    % is scaled inversely proportional to bandwidth we have a simple
    % geometric sum.
    totalTau = tau * (1 - (1/mult)^nscale)/(1-(1/mult));

    % Calculate mean and std dev from tau using fixed relationship
    % between these parameters and tau. See
    % http://mathworld.wolfram.com/RayleighDistribution.html
    EstNoiseEnergyMean = totalTau*sqrt(pi/2);        % Expected mean and
    % values of noise
    EstNoiseEnergySigma = totalTau*sqrt((4-pi)/2);   % Noise threshold
    T = EstNoiseEnergyMean + k*EstNoiseEnergySigma;  % Noise threshold
end
%---- Final computation of key quantities -----

% Orientation - this varies +/- pi/2
or = atan(-sumh2./sumh1);  % Orientation values now range 0 - pi
or(or<0) = or(or<0)+pi;  % Add pi to -ve orientation value so that all

% Quantize to 0 - 180 degrees (for NONMAXSUP)
or = fix(or/pi*180);  % Add pi to -ve orientation value so that all

% Feature type - a phase angle
ft = atan2(sumf,sqrt(sumh1.^2+sumh2.^2));  % Feature type - a phase angle

energy = sqrt(sumf.^2 + sumh1.^2 + sumh2.^2) + epsilon;  % Overall energy

% Compute phase congruency.  The original measure,
% PC = energy/sumAn
% is proportional to the weighted cos(phasedeviation).  This is not very
% localised so this was modified to
% PC = cos(phasedeviation) - |sin(phasedeviation)|
% (Note this was actually calculated via dot and cross products.)  This
% measure
% approximates
% PC = 1 - phasedeviation.
% However, rather than use dot and cross products it is simpler and more
% efficient to simply use acos(energy/sumAn) to obtain the weighted phase
% deviation directly.  Note, in the expression below the noise threshold
% is
% not subtracted from energy immediately as this would interfere with the
% phase deviation computation.  Instead it is subtracted after this
% computation, hence the separate division by sumAn.  Finally this result
% is
% floored at 0, and then weighted for frequency spread.

PC = weight.*max(1 - acos(energy./(sumAn + epsilon)) -
T./(sumAn+epsilon), 0);

%-----------------------------------------------
% CHECKARGS
% Function to process the arguments that have been supplied, assign
% default values as needed and perform basic checks.

function [im, nscale, minWaveLength, mult, sigmaOnf, ...
   k, noiseMethod, cutOff, g] = checkargs(arg)

nargs = length(arg);
if nargs < 1
   error('No image supplied as an argument');
end

% Set up default values for all arguments and then overwrite them
% with with any new values that may be supplied
im = [];
nscale = 5;  % Number of wavelet scales.
minWaveLength = 3;  % Wavelength of smallest scale filter.
mult = 2.1; % Scaling factor between successive filters.
sigmaOnf = 0.65; % Ratio of the standard deviation of the
% Gaussian describing the log Gabor filter's
% transfer function in the frequency domain
% to the filter center frequency.
k = 2.0; % No of standard deviations of the noise
% energy beyond the mean at which we set the
% noise threshold point.
noiseMethod = -1; % Use the median response of smallest scale
% filter to estimate noise statistics
cutOff = 0.5;
g = 10;

% Allowed argument reading states
allnumeric = 1; % Numeric argument values in predefined order
keywordvalue = 2; % Arguments in the form of string keyword
% followed by numeric value
readstate = allnumeric; % Start in the allnumeric state

if readstate == allnumeric
    for n = 1:nargs
        if isa(arg{n},'char')
            readstate = keywordvalue;
            break;
        else
            if     n == 1, im            = arg{n};
            elseif n == 2, nscale        = arg{n};
            elseif n == 3, minWaveLength = arg{n};
            elseif n == 4, mult          = arg{n};
            elseif n == 5, sigmaOnf      = arg{n};
            elseif n == 6, k             = arg{n};
            elseif n == 7, noiseMethod    = arg{n};
            elseif n == 8, cutOff        = arg{n};
            elseif n == 9, g             = arg{n};
            end
        end
    end
end

% Code to handle parameter name - value pairs
if readstate == keywordvalue
    while n < nargs
        if ~isa(arg{n},'char') || ~isa(arg{n+1}, 'double')
            error('There should be a parameter name - value pair');
        end
        if     strncmpi(arg{n},'im'      ,2),      im =
        elseif strncmpi(arg{n},'nscale'  ,2),      nscale =
        elseif strncmpi(arg{n},'minWaveLength',2), minWaveLength =
        end
    end
end
elseif strncmpi(arg{n},'mult',2), mult = arg{n+1};
elseif strncmpi(arg{n},'sigmaOnf',2), sigmaOnf = arg{n+1};
elseif strncmpi(arg{n},'k',1), k = arg{n+1};
elseif strncmpi(arg{n},'noisemethod',3), noiseMethod = arg{n+1};
elseif strncmpi(arg{n},'cutOff',2), cutOff = arg{n+1};
elseif strncmpi(arg{n},'g',1), g = arg{n+1};
else error('Unrecognised parameter name');
end
n = n+2;
if n == nargs
    error('Unmatched parameter name - value pair');
end
end

if isempty(im)
    error('No image argument supplied');
end
if ~isa(im, 'double')
    im = double(im);
end
if nscale < 1
    error('nscale must be an integer >= 1');
end
if minWaveLength < 2
    error('It makes little sense to have a wavelength < 2');
end

%-----------------------------------------------------------------------------
% RAYLEIGHMODE
% Computes mode of a vector/matrix of data that is assumed to come from a
% Rayleigh distribution.
% Usage: rmode = rayleighmode(data, nbins)
% Arguments: data - data assumed to come from a Rayleigh distribution
% nbins - Optional number of bins to use when forming histogram of the data to determine the mode.
% Mode is computed by forming a histogram of the data over 50 bins and then finding the maximum value in the histogram. Mean and standard deviation can then be calculated from the mode as they are related by fixed constants.
% mean = mode * sqrt(pi/2)
% std dev = mode * sqrt((4-pi)/2)
%
% See
% http://mathworld.wolfram.com/RayleighDistribution.html
% http://en.wikipedia.org/wiki/Rayleigh_distribution

10.2 function rmode = rayleighmode(data, nbins)

    if nargin == 1
        nbins = 50;  % Default number of histogram bins to use
    end
    mx = max(data(:));
    edges = 0:mx/nbins:mx;
    n = histc(data(:),edges);
    [dum,ind] = max(n); % Find maximum and index of maximum in histogram
    rmode = (edges(ind)+edges(ind+1))/2;

10.3 function noisevar = evar(y)

    %EVAR Noise variance estimation.
    % Assuming that the deterministic function Y has additive Gaussian noise,
    % EVAR(Y) returns an estimated variance of this noise.
    %
    % Note:
    % ----
    % A thin-plate smoothing spline model is used to smooth Y. It is assumed
    % that the model whose generalized cross-validation score is minimum can
    % provide the variance of the additive noise. A few tests showed that
    % EVAR works very well with "not too irregular" functions.
    %
    % Examples:
    % -------
    % % 1D signal
    % n = 1e6; x = linspace(0,100,n);
    % y = cos(x/10)+(x/50);
    % var0 = 0.02; % noise variance
    % yn = y + sqrt(var0)*randn(size(y));
    % evar(yn) % estimated variance
    %
    % % 2D function
    % [xp,yp] = deal(0:.02:1);
    % [x,y] = meshgrid(xp,yp);
    % f = exp(x+y) + sin((x-2*y)*3);
    % var0 = 0.04; % noise variance
    % fn = f + sqrt(var0)*randn(size(f));
    % evar(fn) % estimated variance
    %
    % % 3D function
%   [x,y,z] = meshgrid(-2:.2:2,-2:.2:2,-2:.2:2);
%   f = x.*exp(-x.^2-y.^2-z.^2);
%   var0 = 0.5; % noise variance
%   fn = f + sqrt(var0)*randn(size(f));
%   evar(fn) % estimated variance
%
% Other examples
% Click <a href="matlab:web('http://www.biomecardio.com/matlab/evar.html')">here</a> for more examples
%
% Note:
% ----
% EVAR is only adapted to evenly-gridded 1-D to N-D data.
% See also VAR, STD, SMOOTHN
%
% -- Damien Garcia -- 2008/04, revised 2009/10

error(nargchk(1,1,nargin));

d = ndims(y);
siz = size(y);
S = zeros(siz);
for i = 1:d
    siz0 = ones(1,d);
siz0(i) = siz(i);
    S = bsxfun(@plus,S,cos(pi*(reshape(1:siz(i),siz0)-1)/siz(i)));
end
S = 2*(d-S);

% N-D Discrete Cosine Transform of Y
if exist('dctn','file')
    DCTy = dctn(y);
else
    error('MATLAB:evar:MissingFunction',...
          'DCTN is required. Download <a href="matlab:web('''',...
          'http://www.biomecardio.com/matlab/dctn.html'">DCTN</a>.')
end
fminbnd(@func,-38,38);

10.4 function score = func(L)

    % Generalized cross validation score
    M = 1-1./(1+10^L*S.^2);
    noisevar = mean(DCTy(:).^2.*M(:).^2);
    score = noisevar/mean(M(:))^2;
end
end
10.5 function \([y, w] = dctn(y, w)\)

% DCTN N-D discrete cosine transform.
% Y = DCTN(X) returns the discrete cosine transform of X. The array Y is
% the same size as X and contains the discrete cosine transform
% coefficients. This transform can be inverted using IDCTN.
% 
% Class Support
% ------------
% Input array can be numeric or logical. The returned array is of class
% double.
% 
% Reference
% --------
% Narasimha M. et al, On the computation of the discrete cosine
% transform, IEEE Trans Comm, 26, 6, 1978, pp 934-936.
% 
% Example
% ------
% RGB = imread('autumn.tif');
% I = rgb2gray(RGB);
% J = dctn(I);
% imshow(log(abs(J)),[]), colormap(jet), colorbar
% 
% The commands below set values less than magnitude 10 in the DCT matrix
% to zero, then reconstruct the image using the inverse DCT.
% 
% J(abs(J)<10) = 0;
% K = idctn(J);
% figure, imshow(I)
% figure, imshow(K,[0 255])
% 
% See also IDCTN, DCT, DCT2.
% 
% -- Damien Garcia -- 2008/06, revised 2009/11

% [Y, W] = DCTN(X, W) uses and returns the weights which are used by the
% program. If DCTN is required for several large arrays of same size, the
% weights can be reused to make the algorithm faster. A typical syntax is
% the following:
% w = [];
% for k = 1:10
% [y{k}, w] = dctn(x{k}, w);
% end
% The weights (w) are calculated during the first call of DCTN then
% reused in the next calls.
% 
error(nargoutchk(1,2,nargin))

y = double(y);
sizey = size(y);
y = squeeze(y);
dimy = ndims(y);
if isvector(y)
    dimy = 1;
    y = y(:);
end

if ~exist('w','var') || isempty(w)
    for dim = 1:dimy
        n = (dimy==1)*numel(y) + (dimy>1)*sizy(dim);
        w(dim) = exp(1i*(0:n-1)'*pi/2/n);
    end
end

if ~isreal(y)
    y = complex(dctn(real(y),w),dctn(imag(y),w));
else
    for dim = 1:dimy
        siz = size(y);
        n = siz(1);
        y = y([1:2:n 2*floor(n/2):-2:2],:);
        y = reshape(y,n,[]);
        y = ifft(y,[],1);
        y = bsxfun(@times,y,w{dim});
        y = real(y);
        y(1,:) = y(1,:)/sqrt(2);
        y = reshape(y,siz);
        y = shiftdim(y,1);
    end
end

y = reshape(y,sizy);

10.6 function blur = blurMetric(original)

% original : entry image

% The idea is from "The Blur Effect: Perception and Estimation with a New No-
% Reference Perceptual Blur Metric"
% Crété-Roffet F., Dolmiere T., Ladret P., Nicolas M. – GRENOBLE – 2007
% In SPIE proceedings – SPIE Electronic Imaging Symposium Conf Human Vision
% and Electronic Imaging, États-Unis d'Amérique (2007)

% Written by DO Quoc Bao, PhD Student in L2TI Laboratory, Paris 13
% University, France
% Email: doquocbao@gmail.com, do.quocbao@l2ti.univ-paris13.fr
% Last changed: 21-09-2008

%%%%%%%%%%%%%%%%%%Note: the output (blur) is in [0,1]; 0 means sharp, 1 means blur%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Please cite the author when you use this code. All remarks are welcome. %
I = double(original);
[y x] = size(I);
Hv = [1 1 1 1 1 1 1 1 1]/9;
Hh = Hv';

B_Ver = imfilter(I,Hv); %blur the input image in vertical direction
B_Hor = imfilter(I,Hh); %blur the input image in horizontal direction

D_F_Ver = abs(I(:,1:x-1) - I(:,2:x)); %variation of the input image (vertical direction)
D_F_Hor = abs(I(1:y-1,:) - I(2:y,:)); %variation of the input image (horizontal direction)

D_B_Ver = abs(B_Ver(:,1:x-1) - B_Ver(:,2:x)); %variation of the blurred image (vertical direction)
D_B_Hor = abs(B_Hor(1:y-1,:) - B_Hor(2:y,:)); %variation of the blurred image (horizontal direction)

T_Ver = D_F_Ver - D_B_Ver; %difference between two vertical variations of 2 image (input and blurred)
T_Hor = D_F_Hor - D_B_Hor; %difference between two horizontal variations of 2 image (input and blurred)

V_Ver = max(0,T_Ver);
V_Hor = max(0,T_Hor);

S_D_Ver = sum(sum(D_F_Ver(2:y-1,2:x-1)));
S_D_Hor = sum(sum(D_F_Hor(2:y-1,2:x-1)));

S_V_Ver = sum(sum(V_Ver(2:y-1,2:x-1)));
S_V_Hor = sum(sum(V_Hor(2:y-1,2:x-1)));

blur_F_Ver = (S_D_Ver-S_V_Ver)/S_D_Ver;
blur_F_Hor = (S_D_Hor-S_V_Hor)/S_D_Hor;
blur = max(blur_F_Ver,blur_F_Hor);

function fr=frequency_ratio(img,Nsub,fcut)
%img input image
%Nsub size of sub region around center point, should be pow of 2
%fcut is the lowpass frequency cutoff

[r c k] = size(img);
rcenter=round(r/2);
ccenter=round(c/2);

simg=(img(round(rcenter-Nsub/2:rcenter+Nsub/2-1),round(ccenter-Nsub/2:ccenter+Nsub/2-1),:));
%compute fft of subimage
simgF=fftshift(fft2(simg));
FM=real(simgF.*conj(simgF));

%subselect low frequency part
fcuti=round(fcut*Nsub);
lpass=sum(sum(FM(Nsub/2-fcuti:Nsub/2+fcuti-1,Nsub/2-fcuti:Nsub/2+fcuti-1)));
sall=sum(sum(FM));
hpass=sall-lpass;
fr=hpass/lpass;

edge_intensity.m
function outval = edge_intensity(img)
% OUTVAL = EDGE_INTENSITY(IMG)
if nargin == 1
    img = double(img);
    % Create horizontal sobel matrix
    w = fspecial('sobel');

    % Get the size of img
    [r c k] = size(img);

    gx = imfilter(img,w,'replicate');
    gy = imfilter(img,w','replicate');

    for m = 1 : r
        for n = 1 : c
            for q = 1 : k
                g(m,n,q) = sqrt(gx(m,n,q)*gx(m,n,q) + gy(m,n,q)*gy(m,n,q));
            end
        end
    end
else
    error('Wrong number of input!');
end

10.8 function [mssim, ssim_map] = ssim(img1, img2, K, window, L)

% ===========================================================================
% SSIM Index with automatic downsampling, Version 1.0
% Copyright(c) 2009 Zhou Wang
% All Rights Reserved.
%
% ===========================================================================
% Permission to use, copy, or modify this software and its documentation
% for educational and research purposes only and without fee is hereby
% granted, provided that this copyright notice and the original authors'
% names appear on all copies and supporting documentation. This program
% shall not be used, rewritten, or adapted as the basis of a commercial
% software or hardware product without first obtaining permission of the
% authors. The authors make no representations about the suitability of

27 February 2014     Motion Imagery Standards Board     35
% This software for any purpose. It is provided "as is" without express
% or implied warranty.

% This is an implementation of the algorithm for calculating the
% Structural SIMilarity (SSIM) index between two images

% Please refer to the following paper and the website with suggested usage
% Z. Wang, A. C. Bovik, H. R. Sheikh, and E. P. Simoncelli, "Image
% quality assessment: From error visibility to structural similarity,"
% IEEE Transactions on Image Processing, vol. 13, no. 4, pp. 600-612,
% http://www.ece.uwaterloo.ca/~z70wang/research/ssim/
% Note: This program is different from ssim_index.m, where no automatic
downsampling is performed. (downsampling was done in the above paper
and was described as suggested usage in the above website.)
% Kindly report any suggestions or corrections to zhouwang@ieee.org

% Input:
% (1) img1: the first image being compared
% (2) img2: the second image being compared
% (3) K: constants in the SSIM index formula (see the above reference).
%     default value: K = [0.01 0.03]
% (4) window: local window for statistics (see the above reference).
%     default window is Gaussian given by
%     window = fspecial('gaussian', 11, 1.5);
% (5) L: dynamic range of the images. default: L = 255

% Output:
% (1) mssim: the mean SSIM index value between 2 images.
%     If one of the images being compared is regarded as
%     perfect quality, then mssim can be considered as the
%     quality measure of the other image.
%     If img1 = img2, then mssim = 1.
% (2) ssim_map: the SSIM index map of the test image. The map
%     has a smaller size than the input images. The actual size
%     depends on the window size and the downsampling factor.

% Basic Usage:
% Given 2 test images img1 and img2, whose dynamic range is 0-255
% [mssim, ssim_map] = ssim(img1, img2);

% Advanced Usage:
% User defined parameters. For example
% K = [0.05 0.05];
% window = ones(8);
% L = 100;
% [mssim, ssim_map] = ssim(img1, img2, K, window, L);
% Visualize the results:
% mssim = -Inf; % Gives the mssim value
% imshow(max(0, ssim_map).^4) % Shows the SSIM index map

if (nargin < 2 || nargin > 5)
    mssim = -Inf;
    ssim_map = -Inf;
    return;
end

if (size(img1) ~= size(img2))
    mssim = -Inf;
    ssim_map = -Inf;
    return;
end

[M N] = size(img1);

if (nargin == 2)
    if ((M < 11) || (N < 11))
        mssim = -Inf;
        ssim_map = -Inf;
        return
    end
    window = fspecial('gaussian', 11, 1.5); %
    K(1) = 0.01; % default settings
    K(2) = 0.03; %
    L = 255;
end

if (nargin == 3)
    if ((M < 11) || (N < 11))
        mssim = -Inf;
        ssim_map = -Inf;
        return
    end
    window = fspecial('gaussian', 11, 1.5);
    L = 255;
    if (length(K) == 2)
        if (K(1) < 0 || K(2) < 0)
            mssim = -Inf;
            ssim_map = -Inf;
            return;
        end
        else
            mssim = -Inf;
            ssim_map = -Inf;
            return;
        end
    end
end

if (nargin == 4)
\[ [H \ W] = \text{size}(\text{window}); \]
\[
\text{if } ((H*W) < 4 \ || \ (H > M) \ || \ (W > N)) \text{ return}
\]
\[
\text{mssim} = -\text{Inf};
\]
\[
\text{ssim\_map} = -\text{Inf};
\]
\[
L = 255;
\]
\[
\text{if } (\text{length}(K) == 2) \text{ return};
\]
\[
\text{if } (K(1) < 0 \ || \ K(2) < 0) \text{ return};
\]
\[
\text{else}
\]
\[
\text{mssim} = -\text{Inf};
\]
\[
\text{ssim\_map} = -\text{Inf};
\]
\[
\text{end}
\]
\[
\text{end}
\]
\[
\text{if } (\text{nargin} == 5) \text{ return};
\]
\[
[H \ W] = \text{size}(\text{window}); \text{ return}
\]
\[
\text{end}
\]
\[
\text{if } (\text{length}(K) == 2) \text{ return};
\]
\[
\text{if } (K(1) < 0 \ || \ K(2) < 0) \text{ return};
\]
\[
\text{else}
\]
\[
\text{mssim} = -\text{Inf};
\]
\[
\text{ssim\_map} = -\text{Inf};
\]
\[
\text{end}
\]
\[
\text{end}
\]
\[
\text{img1} = \text{double}(\text{img1});
\]
\[
\text{img2} = \text{double}(\text{img2});
\]
\[
\% \text{automatic downsampling}
\]
\[
f = \text{max}(1, \text{round}(\text{min}(M,N)/256));
\]
\[
\text{%downsampling by } f
\]
\[
\% \text{use a simple low-pass filter}
\]
\[
\text{if}(f > 1)
\]
\[
\text{lpf} = \text{ones}(f,f);
\]
\[
\text{lpf} = \text{lpf}/\text{sum(lpf(:))};
\]
\[
\text{img1} = \text{imfilter(img1,lpf,'symmetric','same')};
\]
\[
\text{img2} = \text{imfilter(img2,lpf,'symmetric','same')};
\]
\[
\text{img1} = \text{img1}(1:f:end,1:f:end);\]
img2 = img2(1:f:end,1:f:end);
end

C1 = (K(1)*L)^2;
C2 = (K(2)*L)^2;
window = window/sum(sum(window));

mu1 = filter2(window, img1, 'valid');
mu2 = filter2(window, img2, 'valid');
mu1_sq = mu1.*mu1;
mu2_sq = mu2.*mu2;
mul_mu2 = mu1.*mu2;
sigma1_sq = filter2(window, img1.*img1, 'valid') - mu1_sq;
sigma2_sq = filter2(window, img2.*img2, 'valid') - mu2_sq;
sigma12 = filter2(window, img1.*img2, 'valid') - mu1_mu2;

if (C1 > 0 && C2 > 0)
    ssim_map = ((2*mu1_mu2 + C1).*(2*sigma12 + C2))./((mu1_sq + mu2_sq + C1).*(sigma1_sq + sigma2_sq + C2));
else
    numerator1 = 2*mu1_mu2 + C1;
    numerator2 = 2*sigma12 + C2;
    denominator1 = mu1_sq + mu2_sq + C1;
    denominator2 = sigma1_sq + sigma2_sq + C2;
    ssim_map = ones(size(mu1));
    index = (denominator1.*denominator2 > 0);
    ssim_map(index) = (numerator1(index).*numerator2(index))./(denominator1(index).*denominator2(index));
    index = (denominator1 ~= 0) & (denominator2 == 0);
    ssim_map(index) = numerator1(index)./denominator1(index);
end
mssim = mean2(ssim_map);
return

10.9 [Acum Tcum iga]=opticalflow(iraw,iraw_old,ones(size(iraw)),1);

%%% ALIGN TWO FRAMES (f2 to f1)
function[ Acum, Tcum, f2 ] = opticalflow( f1, f2, roi, L )
f2orig = f2;
Acum = [1 0 ; 0 1];
Tcum = [0 ; 0];
for k = L : -1 : 0
    f1d = down( f1, k );
    f2d = down( f2, k );
    ROI = down( roi, k );
    %%% COMPUTE MOTION
    [Fx,Fy,Ft] = spacetime.deriv( f1d, f2d );
    [A,T] = computemotion( Fx, Fy, Ft, ROI );
    %if length(find(isnan(A)))>0
    %keyboard
10.10 function[ A, T ] = computemotion( fx, fy, ft, roi )

function[ A, T ] = computemotion( fx, fy, ft, roi )

% TRIM EDGES
fx = fx( 3:end-2, 3:end-2 );
fy = fy( 3:end-2, 3:end-2 );
ft = ft( 3:end-2, 3:end-2 );
roi = roi( 3:end-2, 3:end-2 );
x = x( 3:end-2, 3:end-2 );
y = y( 3:end-2, 3:end-2 );
ind = find( roi > 0 );
x = x(ind); y = y(ind);
fx = fx(ind); fy = fy(ind); ft = ft(ind);

%x = x.*fx; %fy = y.*fy;
M(1,1) = sum( xfx .* xfx ); M(1,2) = sum( xfx .* yfy ); M(1,3) = sum( xfx .* yfy );
M(1,4) = sum( xfx .* yfy ); M(1,5) = sum( xfx .* yfy ); M(1,6) = sum( xfx .* yfy );
M(2,1) = M(1,2); M(2,2) = sum( yfx .* yfx ); M(2,3) = sum( yfx .* yfx );
M(2,4) = sum( yfx .* yfx ); M(2,5) = sum( yfx .* yfy ); M(2,6) = sum( yfx .* yfy );
M(3,1) = M(3,2); M(3,3) = sum( xfy .* xfy ); M(3,4) = sum( xfy .* xfy );
M(3,5) = sum( xfy .* xfy ); M(3,6) = sum( xfy .* xfy );
M(4,1) = M(1,4); M(4,2) = M(2,4); M(4,3) = M(3,4);
M(4,4) = sum( yfy .* yfy ); M(4,5) = sum( yfy .* yfy ); M(4,6) = sum( yfy .* yfy );
M(5,1) = M(1,5); M(5,2) = M(2,5); M(5,3) = M(3,5);
M(5,4) = M(4,5); M(5,5) = sum( fx .* fx ); M(5,6) = sum( fx .* fx );
M(6,1) = M(1,6); M(6,2) = M(2,6); M(6,3) = M(3,6);
M(6,4) = M(4,6); M(6,5) = M(5,6); M(6,6) = sum( fy .* fy );

k = ft + xfx + yfy;
b(1) = sum( k .* xfx ); b(2) = sum( k .* yfy );
b(3) = sum( k .* xfy ); b(4) = sum( k .* yfy );
b(5) = sum( k .* fx ); b(6) = sum( k .* fy );
v = inv(M) * b';
A = [v(1) v(2) ; v(3) v(4)];
T = [v(5) ; v(6)];

10.11 function[ A2, T2 ] = accumulatetrap( Acum, Tcum, A, T )

A2 = A * Acum;
T2 = A*Tcum + T;

%%% ACCUMULATE WARPS
10.12 Vertical blocking detector code:

```matlab
macroAngle = 0; % to detect vertical lines
e = edge(iraw,.005); % extract edges
se = strel('line',7,macroAngle);
er = imerode(e,se); % throw away horizontal lines
ers = sum(er); % sum columns
ibufl = [ibufl ers]; % store in rolling buffer
if length(ibufl) > 8192
    ibufl = ibufl(length(ers)+1:length(ibufl)); % keyboard
N = length(ibufl);
pxx = pwelch(ibufl,ones(1,2048)/2048); % compute spectrum
pxxd = detrend(10*log10(pxx));
Np = length(pxxd);
ibin = round(2*Np/16)+1;
idx = ibin;
mbe = (sum(pxxd(idx)))/length(idx); % macroblock energy
idx1 = round(ibin+.04*Np):round(ibin+.05*Np);
fe = sum(pxxd(idx1))/length(idx1); % average floor energy
rv = (mbe - fe)); % difference in log = ratio
end;
```

10.13 Horizontal blocking detector code:

```matlab
macroAngle = 90; % to detect horizontal lines
se = strel('line',7,macroAngle);
e = edge(iraw,.005); % compute edges
er = imerode(e,se); % throw away vertical lines
ers2 = sum(er'); % sum rows
ibufl = [ibufl2 ers]; % store in rolling buffer

if length(ibufl2) > 8192
    ibuf2 = ibuf2(length(ers)+1:length(ibufl2)); % keyboard
N = length(ibufl2);
pxx = pwelch(ibufl2,ones(1,2048)/2048); % compute spectrum
pxxd = detrend(10*log10(pxx));
Np = length(pxxd);
ibin = round(2*Np/16)+1;
idx = ibin;
mbe = (sum(pxxd(idx)))/length(idx); % macro block energy
idx1 = round(ibin+.04*Np):round(ibin+.05*Np);
fe = sum(pxxd(idx1))/length(idx1); % average floor energy
rh = (mbe - fe)); % difference in log = ratio
end;
```
10.14 function RER=perceptual_RER(iraw)

function RER=perceptual_RER(iraw)
%function takes input an intensity image
%and computes the Relative Edge Response
%estimate of the slope
%of the line spread function.

%estimate RER on original image
r1=RERpoly(iraw);
%downsample and repeat
r2=RERpoly(blur(iraw,1));
%use ratio to make non-linear adjustment

r=r1/r2;
RER=(r1*(2/r)^3).^(2/r);

function rer=RERpoly(iraw)
PSF = fspecial('gaussian',10,10); %define a blurring function
iraw2=imfilter(iraw,PSF,'conv'); %smooth the input image
icon=phasecongmono(iraw2); %compute phase congruency image
se=strel('square',3); %parameter for erode
iconc=imerode(icon,se); %reduce false hits
icond=imdilate(iconc,se); %re-enlarge
i2=iraw(1:size(icond,1),1:size(icond,2)); %select original pixels
ym=icond.*double(i2); %convert to double
ymz=sort(ym(:)); %sort intensities
idx=find(ymz); %find non-zero entries
ymzz=ymz(idx); %select non-zero entries
idx=find(ymzz==255); %remove repeating entries
yms=ymzz(idx); %select non-repeating limit entries
[ymin xmini]=min(yms); %find min and location
[ymax xmaxi]=max(yms); %find max and location
xwi=xmaxi-xmini; %calculate width
xci=xmini+round(xwi/2); %calculate center
ib=xci-round(xwi/4); %select interval for linear fit
ie=xci+round(xwi/4); %
p=polyfit(ib:ie,yms(ib:ie)',1); %perform fit
rer=p(1)*length(ib:ie)/2.5; %estimate RER

10.15% RES = blur(IM, LEVELS, FILT)

% Blur an image, by filtering and downsampling LEVELS times
% (default=1), followed by upsampling and filtering LEVELS times. The
% blurring is done with filter kernel specified by FILT (default =
% 'binom5'), which can be a string (to be passed to namedFilter), a
% vector (applied separably as a 1D convolution kernel in X and Y), or
% a matrix (applied as a 2D convolution kernel). The downsampling is
% always by 2 in each direction.

% Eero Simoncelli, 3/04.

function res = blur(im, nlevs, filt)

%------------------------------------------------------------
% OPTIONAL ARGS:

if (exist('nlevs') ~= 1)
    nlevs = 1;
end

if (exist('filt') ~= 1)
    filt = 'binom5';
end

%------------------------------------------------------------

if isstr(filt)
    filt = namedFilter(filt);
end

filt = filt/sum(filt(:));

if nlevs > 0
    if (any(size(im)==1))
        if (~any(size(filt)==1))
            error('Cant apply 2D filter to 1D signal');
        end
        if (size(im,2)==1)
            filt = filt(:);
        else
            filt = filt(:)';
        end

        in = corrDn(im,filt,'reflect1',(size(im)==1)+1);
        out = blur(in, nlevs-1, filt);
        res = upConv(out, filt, 'reflect1', (size(im)==1)+1, [1 1], size(im));
    elseif (any(size(filt)==1))
        filt = filt(:);

        in = corrDn(im,filt,'reflect1',[2 1]);
        in = corrDn(in,filt','reflect1',[1 2]);
        out = blur(in, nlevs-1, filt);
        res = upConv(out, filt', 'reflect1', [1 2], [1 1],
                    [size(out,1),size(im,2)]);
        res = upConv(res, filt, 'reflect1', [2 1], [1 1], size(im));
    else
        in = corrDn(im,filt,'reflect1',size(im));
        out = blur(in, nlevs-1, filt);
        res = upConv(out, filt, 'reflect1', [1 2], [1 1],
                    [size(out,1),size(im,2)]);
        res = upConv(res, filt, 'reflect1', [2 1], [1 1], size(im));
    end
else
    in = corrDn(im,filt,'reflect1',size(im));
    out = blur(in, nlevs-1, filt);
    res = upConv(out, filt, 'reflect1', [1 2], [1 1],
                  [size(out,1),size(im,2)]);
    res = upConv(res, filt, 'reflect1', [2 1], [1 1], size(im));
end
else
  in = corrDn(im,filt,'reflect1',[2 2]);
  out = blur(in, nlevs-1, filt);
  res = upConv(out, filt, 'reflect1', [2 2], [1 1], size(im));
end
else
  res = im;
end

\[ 10.16\% \text{RES} = \text{upConv(}\text{IM, FILT, EDGES, STEP, START, STOP, RES}\text{)} \]

\begin{verbatim}
function result = upConv(im,filt,edges,step,start,stop,res)
  \% UPCONV is a function that upsample matrix IM, followed by convolution with matrix FILT. These
  \% arguments should be 1D or 2D matrices, and IM must be larger (in both dimensions) than FILT. The origin of filt
  \% is assumed to be floor(size(filt)/2)+1.
  \% EDGES is a string determining boundary handling:
  \%  'circular' - Circular convolution
  \%  'reflect1' - Reflect about the edge pixels
  \%  'reflect2' - Reflect, doubling the edge pixels
  \%  'repeat' - Repeat the edge pixels
  \%  'zero' - Assume values of zero outside image boundary
  \%  'extend' - Reflect and invert
  \%  'dont-compute' - Zero output when filter overhangs OUTPUT boundaries
  \% Upsampling factors are determined by STEP (optional, default=[1 1]), a 2-vector [y,x].
  \% The window over which the convolution occurs is specified by START
  \% (optional, default=[1,1], and STOP (optional, default = step .* (size(IM) + floor((start-1)./step))).
  \% RES is an optional result matrix. The convolution result will be destructively added into this matrix. If this argument is passed, the result matrix will not be returned. DO NOT USE THIS ARGUMENT IF YOU DO NOT UNDERSTAND WHAT THIS MEANS!!
  \% NOTE: this operation corresponds to multiplication of a signal vector by a matrix whose columns contain copies of the time-reversed (or space-reversed) FILT shifted by multiples of STEP. See corrDn.m for the operation corresponding to the transpose of this matrix.

  \% Eero Simoncelli, 6/96. revised 2/97.

  function result = upConv(im,filt,edges,step,start,stop,res)
  \% THIS CODE IS NOT ACTUALLY USED! (MEX FILE IS CALLED INSTEAD)

  fprintf(1,'WARNING: You should compile the MEX version of "upConv.c", \n  found in the MEX subdirectory of matlabPyrTools, and put it in your matlab path. It is MUCH faster, and provides more boundary-handling options.\n');
\end{verbatim}
%% OPTIONAL ARGS:

if (exist('edges') == 1)
    if (strcmp(edges, 'reflect1') ~= 1)
        warning('Using REFLECT1 edge-handling (use MEX code for other options).');
    end
end

if (exist('step') ~= 1)
    step = [1,1];
end

if (exist('start') ~= 1)
    start = [1,1];
end

% A multiple of step
if (exist('stop') ~= 1)
    stop = step .* (floor((start-ones(size(start)))./step)+size(im))
end

if (ceil((stop(1)+1-start(1)) / step(1)) ~= size(im,1) )
    error('Bad Y result dimension');
end
if (ceil((stop(2)+1-start(2)) / step(2)) ~= size(im,2) )
    error('Bad X result dimension');
end

if (exist('res') ~= 1)
    res = zeros(stop-start+1);
end

%------------------------------------------------------------
tmp = zeros(size(res));
result = rconv2(tmp,filt) + res;

10.17% RES = corrDn(IM, FILT, EDGES, STEP, START, STOP)

% Compute correlation of matrices IM with FILT, followed by downsampling. These arguments should be 1D or 2D matrices, and IM must be larger (in both dimensions) than FILT. The origin of filt is assumed to be floor(size(filt)/2)+1.
% EDGES is a string determining boundary handling:
% 'circular' - Circular convolution
% 'reflect1' - Reflect about the edge pixels
% 'reflect2' - Reflect, doubling the edge pixels
% 'repeat' - Repeat the edge pixels
% 'zero' - Assume values of zero outside image boundary
% 'extend' - Reflect and invert (continuous values and derivs)
% 'dont-compute' - Zero output when filter overhangs input boundaries
% Downsampling factors are determined by STEP (optional, default=[1 1]),
% which should be a 2-vector [y,x].
% The window over which the convolution occurs is specified by START
% (optional, default=[1,1], and STOP (optional, default=size(IM)).
% NOTE: this operation corresponds to multiplication of a signal
% vector by a matrix whose rows contain copies of the FILT shifted by
% multiples of STEP. See upConv.m for the operation corresponding to
% the transpose of this matrix.

% Eero Simoncelli, 6/96, revised 2/97.

function res = corrDn(im, filt, edges, step, start, stop)
%
%% NOTE: THIS CODE IS NOT ACTUALLY USED! (MEX FILE IS CALLED INSTEAD)

fprintf(1,'WARNING: You should compile the MEX version of "corrDn.c",\nfound in the MEX subdirectory of matlabPyrTools, and put it in your matlab
path. It is MUCH faster, and provides more boundary-handling options.\n');

% OPTIONAL ARGS:
if (exist('edges') == 1)
    if (strcmp(edges,'reflect1') ~= 1)
        warning('Using REFLECT1 edge-handling (use MEX code for other
options.'););
    end
end

if (exist('step') ~= 1)
    step = [1,1];
end

if (exist('start') ~= 1)
    start = [1,1];
end

if (exist('stop') ~= 1)
    stop = size(im);
end

% Reverse order of taps in filt, to do correlation instead of convolution
filt = filt(size(filt,1):-1:1,size(filt,2):-1:1);

tmp = rconv2(im,filt);
### 10.18 Function

**probabilityVector=multinomialLogisticProbability(beta,x)**

- Returns probabilities for each class
- Supports matrix of observations `x` of size `n` observations by `p` predictor variables
- `beta` is matrix of `p` x `(k-1)` coefficients (see table)

```matlab
k = size(beta,2) + 1;
[n,p] = size(x);
pstar = size(beta,1);
eta = repmat(beta(1,:),n,1) + x*beta(2:pstar,:);
pi = [exp(eta) ones(n,1)];
pi = pi ./ repmat(sum(pi,2),1,k);
probabilityVector=pi;
```

---

### 10.19 Function

**function [fx, fy, ft] = spacetimederiv(f1, f2)**

```matlab
f1=double(f1);

%%% DERIVATIVE FILTERS
pre = [0.5 0.5];
deriv = [0.5 -0.5];

%%% SPACE/TIME DERIVATIVES
fpt = pre(1)*f1 + pre(2)*f2;  % pre-filter in time
fdt = deriv(1)*f1 + deriv(2)*f2;  % differentiate in time
fx = conv2( conv2( fpt, pre', 'same' ), deriv, 'same' );
fy = conv2( conv2( fpt, pre, 'same' ), deriv', 'same' );
ft = conv2( conv2( fdt, pre', 'same' ), pre, 'same' );
```

---

### 10.20 Function

**function [f] = down(f, L);**

```matlab
blur = [1 2 1]/4;
for k = 1 : L
    f = conv2( conv2( f, blur, 'same' ), blur', 'same' );
    f = f(1:2:end,1:2:end);
end
```
11 Appendix

Table 3: PSNR Coefficient 1-35 (from psnrcofs002.csv)

<table>
<thead>
<tr>
<th>row</th>
<th>coeff</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>79.726</td>
</tr>
<tr>
<td>2</td>
<td>-911.63</td>
</tr>
<tr>
<td>3</td>
<td>-132.44</td>
</tr>
<tr>
<td>4</td>
<td>-0.75818</td>
</tr>
<tr>
<td>5</td>
<td>1761.3</td>
</tr>
<tr>
<td>6</td>
<td>-27.805</td>
</tr>
<tr>
<td>7</td>
<td>-0.40457</td>
</tr>
<tr>
<td>8</td>
<td>0.026766</td>
</tr>
<tr>
<td>9</td>
<td>4.1022</td>
</tr>
<tr>
<td>10</td>
<td>-7.1777</td>
</tr>
<tr>
<td>11</td>
<td>-0.34222</td>
</tr>
<tr>
<td>12</td>
<td>0.19063</td>
</tr>
<tr>
<td>13</td>
<td>5876.2</td>
</tr>
<tr>
<td>14</td>
<td>6.0383</td>
</tr>
<tr>
<td>15</td>
<td>7894.5</td>
</tr>
<tr>
<td>16</td>
<td>-397.57</td>
</tr>
<tr>
<td>17</td>
<td>-0.44703</td>
</tr>
<tr>
<td>18</td>
<td>-1.4141</td>
</tr>
<tr>
<td>19</td>
<td>388.48</td>
</tr>
<tr>
<td>20</td>
<td>478.93</td>
</tr>
<tr>
<td>21</td>
<td>-2.0242</td>
</tr>
<tr>
<td>22</td>
<td>-8.7549</td>
</tr>
<tr>
<td>23</td>
<td>0.58589</td>
</tr>
<tr>
<td>24</td>
<td>-2330.1</td>
</tr>
<tr>
<td>25</td>
<td>73.647</td>
</tr>
<tr>
<td>26</td>
<td>2.0376</td>
</tr>
<tr>
<td>27</td>
<td>-1.467</td>
</tr>
<tr>
<td>28</td>
<td>-152.57</td>
</tr>
<tr>
<td>29</td>
<td>-43.172</td>
</tr>
<tr>
<td>30</td>
<td>0.029247</td>
</tr>
<tr>
<td>31</td>
<td>0.83946</td>
</tr>
<tr>
<td>32</td>
<td>-25.51</td>
</tr>
<tr>
<td>33</td>
<td>0.74037</td>
</tr>
<tr>
<td>34</td>
<td>0.001745</td>
</tr>
<tr>
<td>35</td>
<td>-0.00336</td>
</tr>
</tbody>
</table>
PSNR Coefficient 36-67
(from psnrcofs002.csv)

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>36</td>
<td>-0.34324</td>
</tr>
<tr>
<td>37</td>
<td>-0.27475</td>
</tr>
<tr>
<td>38</td>
<td>-0.00644</td>
</tr>
<tr>
<td>39</td>
<td>0.003327</td>
</tr>
<tr>
<td>40</td>
<td>-146.68</td>
</tr>
<tr>
<td>41</td>
<td>1.8708</td>
</tr>
<tr>
<td>42</td>
<td>-13.493</td>
</tr>
<tr>
<td>43</td>
<td>135.09</td>
</tr>
<tr>
<td>44</td>
<td>-537.89</td>
</tr>
<tr>
<td>45</td>
<td>-9.9933</td>
</tr>
<tr>
<td>46</td>
<td>7.6324</td>
</tr>
<tr>
<td>47</td>
<td>-0.17343</td>
</tr>
<tr>
<td>48</td>
<td>0.20134</td>
</tr>
<tr>
<td>49</td>
<td>10.095</td>
</tr>
<tr>
<td>50</td>
<td>11.668</td>
</tr>
<tr>
<td>51</td>
<td>0.13275</td>
</tr>
<tr>
<td>52</td>
<td>-0.25098</td>
</tr>
<tr>
<td>53</td>
<td>0.002964</td>
</tr>
<tr>
<td>54</td>
<td>-0.13687</td>
</tr>
<tr>
<td>55</td>
<td>-0.06155</td>
</tr>
<tr>
<td>56</td>
<td>0.007019</td>
</tr>
<tr>
<td>57</td>
<td>0.002278</td>
</tr>
<tr>
<td>58</td>
<td>0.3511</td>
</tr>
<tr>
<td>59</td>
<td>0.12827</td>
</tr>
<tr>
<td>60</td>
<td>-0.00526</td>
</tr>
<tr>
<td>61</td>
<td>-0.00522</td>
</tr>
<tr>
<td>62</td>
<td>-3.4227</td>
</tr>
<tr>
<td>63</td>
<td>0.14968</td>
</tr>
<tr>
<td>64</td>
<td>0.2767</td>
</tr>
<tr>
<td>65</td>
<td>0.00397</td>
</tr>
<tr>
<td>66</td>
<td>0.18427</td>
</tr>
<tr>
<td>67</td>
<td>0.006018</td>
</tr>
</tbody>
</table>