VTI — Technical documentation
Version 9.00
Created 2021-10-18

www.vti.se

Mean Profile Depth

A Reference Implementation

Peter Andrén
Thomas Lundberg

vti
FINDING A BETTER WAY

Preface

This technical documentation presents a proposal for a reference implementation of the "ISO 13473-1:2019(E) — Corrected version 2021-06" standard "Characterization of pavement texture by use of surface profiles — Part 1: Determination of mean profile depth".

The main intention with this work is to establish a common implementation of the Mean Profile Depth. The Matlab[1] code can be used as it is, or, preferably, as a benchmark to other implementations. (Please report any bugs found to `peter.andren@vti.se`.)

The text below assume some basic knowledge of programming. All expressions related to programming are set with `typewriter text`. Definitions of the terms and concepts are given in the standard.

The implementation of the Mean Profile Depth presented here has its origin in the technical qualification tests that VTI conducts in the Swedish Transport Administration's procurement of road surface assessment, where we need to calculate the reference MPDs from a millimeter texture profile.

Linköping, October 2021

Peter Andrén and Thomas Lundberg

[1] Matlab is a registered trademark of The MathWorks, Inc.

Contents

1 The `mean_profile_depth.m` function

The input arguments to the `mean_profile_depth` function are a mandatory texture profile and an optional `struct` with options. Dropouts in the texture profile must be represented with `NaN`s. The function can handle only one profile at the time.

Without the options `struct` all defaults are used. The return variable is a result `struct`. A typical Matlab session would look something like this:

```
>> text_prof = load('texture_data.txt');
>> MPD = mean_profile_depth(text_prof);
>> MPD
MPD =
  struct with fields:

        MSD: [1×10000 double]
        MPD: [1×50 double]
        STD: [1×50 double]
```

If we set `OPT.verbose = true` the function will present some information on the processing:

```
>> text_prof = load('texture_data.txt');
>> OPT.verbose = true;
>> MPD = mean_profile_depth(text_prof, OPT);

Running 'mean_profile_depth' with:
OPT.verbose                    true
OPT.profile_dx                 1.00 mm
OPT.resample_to_dx             0.00 mm
OPT.spot_measurement           false
OPT.evaluation_length          20 m
OPT.segment_length             100 mm
OPT.extreme_MSD_remove         false
OPT.calculate_EDT              false

7.5:  Spike removal --- 6347 samples affected
7.5:  Interpolation and nearest neighbor ...
       extrapolation of the spikes.
7.6:  Adding mirrored segments.
7.6:  Removal of long-wavelength components ...
       (high-pass filtering).
7.6:  Normalization of profile sharpness ...
       (low-pass filtering).
7.6:  Removing mirrored segments.
7.8:  Calculate the MSDs.
7.5:  Set MSDs with more than 5 % spikes to invalid.
7.10: Averaging of MSD to determine the MPD.
```

First, all options are presented. In this case the default values are used (except for `OPT.verbose` obviously). Next, the processing steps are presented with chapter numbers to the standard.

The options are:

`OPT.profile_dx`: The sample distance of the texture profile. This can be any value up to 1 millimeter. If the sample distance is not 1 or 0.5 millimeter resampling is needed.

`OPT.resample_to_dx` For resampling from longer to shorter sample distance (from, say, 1 mm or 0.8 mm to 0.5 mm) linear interpolation is used. For resampling from shorter to longer sample distance "binning" is used (as stated in the standard: "Calculate the arithmetic average of all samples that fall within the required spacing.")

`OPT.spot_measurement` For spot measurement the older MPD-filtering with slope suppression is used.

`OPT.evaluation_length` Also known as presentation length. Can be anything from 1.0 meter.

`OPT.extreme_MSD_remove` and `OPT.calculate_EDT` are given as "optional" in the standard. Both are `false` by default. `OPT.extreme_MSD_remove = true` will run the three point moving median over the MSD values before the averaging to MPD (according to chapter 7.9 "Extreme MSD value removal" in the standard).

`OPT.calculate_EDT = true` will calculate the estimated texture depth (EDT) and export the results to the results `struct`.

2 MPD calculated from the VTI test profiles

We have used eight texture profiles to test the MPD implementation. The first seven profiles are taken from the VTI Road Surface Tester and have a 1 millimeter sample distance. These seven profiles all have 1 000 000 samples, making them 1 000 meters long each. The eighth profile is taken from a newer reference equipment, which is currently under development at VTI. This equipment produces a profile with an about 0.234 mm sample distance (0.234337194819742 to make the profile exactly 1 000 meters). A zip-file (`texture_profiles.zip`) can be downloaded from `https://www.erpug.org/index.php?contentID=239`.

The table below gives some data on the test profiles, including the MPD as calculated by the `mean_profile_depth.m` function. MPD values at a 20 meter evaluation length can be found in chapter 3.

Object	Location	MPD	Pavement
1	Tjällmo	2.486563	Surface dressing, 11 mm
2	Kätkesuando	0.739969	Chipseal
3	Huskvarna	1.688548	Porous asphalt, 16 mm
4	Sturefors	1.178962	Stone mastic asphalt, 16 mm
5	Gistad	0.649049	Stone mastic asphalt, remix 16 mm
6	Linghem	0.417096	Dense asphalt concrete, 11 mm
7	Bestorp	1.314730	Chipseal
8	Nykil	0.806073	Dense asphalt concrete, 11 mm

The code to get the MPD values in the table above is:

```
for obj = 1:8
  MM{obj} = load(sprintf('obj%02d.mm', obj));
end

MPD = mean_profile_depth(MM{1}); nanmean(MPD.MSD)
MPD = mean_profile_depth(MM{2}); nanmean(MPD.MSD)
MPD = mean_profile_depth(MM{3}); nanmean(MPD.MSD)
MPD = mean_profile_depth(MM{4}); nanmean(MPD.MSD)
MPD = mean_profile_depth(MM{5}); nanmean(MPD.MSD)
MPD = mean_profile_depth(MM{6}); nanmean(MPD.MSD)
MPD = mean_profile_depth(MM{7}); nanmean(MPD.MSD)
OPT.profile_dx              = 0.234337194819742;
OPT.resample_to_dx          = 1.00;
MPD = mean_profile_depth(MM{8}, OPT); nanmean(MPD.MSD)
```

Observe that you will get slightly different results if you substitute `nanmean(MPD.MSD)` with `nanmean(MPD.MPD)` in the code above. In `nanmean(MPD.MSD)` all `NaN`s will effectively be replaced with the mean value of all non-`NaN`s in the vector, whereas in `nanmean(MPD.MPD)` the `NaN`s will effectively be replaced at the evaluation length level. So, these numbers can differ a little. However, the differences from the test profiles are very small.

3 MPD values at a 20 meter evaluation length

The MPD values presented below are calculated from the eight reference texture profiles. Profiles 1–7 have a 1.0 mm sampling distance. The eighth profile has a 0.234337194819742 mm sampling distance, and, thus, must be resampled to either 1.0 or 0.5 mm. See chapter 2 for more information on the texture profiles.

Distance	1	2	3	4	5	6	7	8/1.0 mm	8/0.5 mm
20	2.489426	0.749919	1.026298	1.183043	0.745484	0.786424	1.487803	0.900217	0.911247
40	2.338584	0.737039	1.162250	1.285744	0.612833	0.354284	1.552341	0.970928	0.984036
60	2.489438	0.639774	1.383389	1.250810	0.479793	0.328037	1.518877	1.060355	1.070860
80	2.522691	0.732028	1.368255	1.137396	0.494120	0.385800	1.343546	0.753472	0.764133
100	2.524073	0.783037	1.599533	1.263560	0.559962	0.313114	1.436786	0.903924	0.918567
120	2.575476	0.671848	1.680689	1.191170	0.717918	0.384797	1.409217	1.045314	1.055008
140	2.420716	0.694479	1.734794	1.198741	0.684273	0.257081	1.416851	0.841128	0.856164
160	2.401482	0.723648	1.852644	1.141901	0.759803	0.229901	1.483204	1.002591	1.018847
180	2.414924	0.713958	1.713355	1.230508	0.803846	0.281827	1.554286	0.706074	0.715138
200	2.384392	0.688900	1.692981	1.154633	0.775434	0.276298	1.173432	0.695894	0.708876
220	2.420992	0.725872	1.715914	1.098160	0.797973	0.316630	1.064742	0.794542	0.806912
240	2.540650	0.739742	1.804677	1.149419	0.521648	0.317551	1.231348	0.725953	0.736313
260	2.482858	0.710721	1.769678	1.226227	0.420398	0.367964	1.144020	0.983874	0.996138
280	2.440926	0.657330	1.770909	1.252083	0.480628	0.461460	1.270660	0.687386	0.698504
300	2.437382	0.683481	1.780510	1.244290	0.479884	0.482359	1.281531	0.927210	0.937595
320	2.370876	0.696855	1.756233	1.200027	0.688962	0.415295	1.285171	0.808365	0.822463
340	2.383269	0.700084	1.705840	1.218166	0.626481	0.510099	1.310896	0.837888	0.852016
360	2.419194	0.682884	1.741716	1.031337	0.653224	0.493007	1.352073	1.172816	1.187918
380	2.464501	0.702669	1.631486	1.115523	0.553437	0.454754	1.280794	0.824355	0.837959
400	2.178633	0.706961	1.691529	1.248152	0.575330	0.487305	1.186119	0.896920	0.911404
420	2.359456	0.772705	1.734819	1.273515	0.511083	0.404725	1.237479	0.847456	0.859249
440	2.538140	0.718667	1.841530	1.150063	0.619097	0.457804	1.242794	0.789208	0.803778
460	2.465947	0.823858	1.956754	1.159967	0.758423	0.436497	1.274638	0.755746	0.766883
480	2.555476	0.822582	1.827375	1.240265	0.805115	0.416977	1.161259	0.705810	0.716633
500	2.428599	0.818398	1.899218	1.129076	0.963105	0.441955	1.197097	0.839571	0.851913
520	2.501602	0.761249	1.868245	1.289444	0.869791	0.532035	1.230121	0.626908	0.637995
540	2.588193	0.736513	1.790898	1.096615	0.773882	0.556279	1.260508	0.795616	0.807761
560	2.584645	0.704929	1.871595	1.274733	0.795108	0.353566	1.196401	0.671495	0.684790
580	2.563280	0.767308	1.824863	1.122757	0.842385	0.332329	1.322420	0.825102	0.836427
600	2.552870	0.850607	1.781389	1.140004	0.647407	0.337992	1.433120	0.734676	0.746011
620	2.811339	0.794659	1.749321	1.244850	0.561892	0.269825	1.559899	0.924041	0.939551
640	2.808563	0.731888	1.726153	1.161967	0.769644	0.340262	1.494937	0.746491	0.759580
660	2.740646	0.798984	1.684389	1.350292	0.644499	0.387872	1.340813	0.750726	0.761837
680	2.673048	0.812697	1.727469	1.362027	0.620572	0.418536	1.280588	0.836190	0.848982
700	2.367248	0.783280	1.661966	1.292221	0.517810	0.392489	1.335371	0.855471	0.867511
720	2.347956	0.780608	1.712134	1.248179	0.579397	0.495164	1.272419	0.747926	0.762080
740	2.531391	0.743477	1.584364	0.948040	0.609886	0.481483	1.209891	0.912955	0.925068
760	2.491101	0.735200	1.695133	1.219252	0.388721	0.428465	1.264117	0.773271	0.783828
780	2.605110	0.740148	1.762119	1.127539	0.560392	0.384317	1.041320	0.633410	0.644639
800	2.578155	0.763362	1.734788	1.111577	0.682079	0.360754	1.208060	0.669461	0.679885
820	2.489235	0.765786	1.684671	1.122962	0.680642	0.342239	1.180012	0.887759	0.898321
840	2.476254	0.739295	1.601567	0.952255	0.827582	0.386781	1.109171	0.881784	0.892854
860	2.451318	0.736911	1.756961	0.989801	0.579011	0.430423	1.488560	0.832874	0.844886
880	2.374910	0.813711	1.579459	1.022410	0.567359	0.500363	1.574782	0.808862	0.820204
900	2.407869	0.763509	1.613313	1.058111	0.546767	0.533472	1.677572	0.740207	0.752935
920	2.483117	0.731598	1.637503	1.256237	0.549910	0.544843	1.510063	0.609873	0.618581
940	2.382256	0.677634	1.673819	1.090854	0.522646	0.487369	1.350236	0.648062	0.657653
960	2.376013	0.690881	1.786618	1.275520	0.550067	0.512403	1.169182	0.595878	0.604283
980	2.525342	0.727315	1.636429	1.200609	0.776608	0.501759	1.124323	0.568441	0.577916
1000	2.589724	0.749474	1.546442	1.223097	0.902134	0.481857	1.205629	0.749168	0.761880

4 The Matlab code of the `mean_profile_depth.m` function

```
function RES = mean_profile_depth(TX, OPT)
% RES = mean_profile_depth(TX, OPT)
%
% Version 1.0: 2020-03-09.  First public version.
% Version 1.1: 2021-01-27.  Minor changes, mainly to text comments.
% Version 1.2: 2021-10-18.  Remove mirroring option (now mandatory).
%                           Make the spkie identification code look more like the text in the standard.
%
% 'TX' is the texture profile.
%
% 'OPT' is the options structure.  The document "Mean Profile Depth -- A Reference Implementation" will give you the details.
%
% Default values for the options:
% OPT.verbose                   = false;
% OPT.profile_dx                = 1.00;
% OPT.resample_to_dx            = 0.00;
% OPT.spot_measurement          = false;
% OPT.evaluation_length         = 20;
% OPT.extreme_MSD_remove        = false;
% OPT.calculate_EDT             = false;
%
% Copyright Peter Andrén, VTI, Sweden (peter.andren@vti.se and www.vti.se)
%
% DISCLAIMER The code is provided "as is" without warranty of any kind.

% No options structure given -- all defaults used.
if(nargin == 1)
  OPT.verbose = false;
end

% The function handles only one texture profile at the time.
if(min(size(TX)) > 1)
  error('Only one texture profile at the time please!');
end

% Force the texture profile to be a column vector.
TX = TX(:);

% Default is quiet mode.
if(~isfield(OPT, 'verbose'))
  OPT.verbose = false;
end

% Default is a texture profile with a one millimeter sampling distance.
if(~isfield(OPT, 'profile_dx'))
  OPT.profile_dx = 1.00;
end

% Re-sample to this sampling distance.  Default is to not resample the profile.
if(~isfield(OPT, 'resample_to_dx'))
  OPT.resample_to_dx = 0.0;
end

% Spot measurements use slope suppression, as 'filtfilt' is not recommended on very short sections.
```

```
if(~isfield(OPT, 'spot_measurement'))
  OPT.spot_measurement = false;
end

% The evaluation length (aka presentation length or baselength) in meters.  (20 meters is commonly used in Sweden, hence the default.)
if(~isfield(OPT, 'evaluation_length'))
  OPT.evaluation_length = 20;
end

% Extreme MSD value removal (optional).
if(~isfield(OPT, 'extreme_MSD_remove'))
  OPT.extreme_MSD_remove = false;
end

% Calculates the EDT (optional).
if(~isfield(OPT, 'calculate_EDT'))
  OPT.calculate_EDT = false;
end

if(OPT.verbose == true)
  fprintf('\nRunning ''mean_profile_depth'' with:\n');
  fprintf('%-30s%s\n',           'OPT.verbose', mat2str(OPT.verbose));
  fprintf('%-30s%0.2f mm\n',     'OPT.profile_dx', OPT.profile_dx);
  fprintf('%-30s%0.2f mm\n',     'OPT.resample_to_dx', OPT.resample_to_dx);
  fprintf('%-30s%s\n',           'OPT.spot_measurement', mat2str(OPT.spot_measurement));
  fprintf('%-30s%d m\n',         'OPT.evaluation_length', OPT.evaluation_length);
  fprintf('%-30s%s\n',           'OPT.extreme_MSD_remove', mat2str(OPT.extreme_MSD_remove));
  fprintf('%-30s%s\n\n',         'OPT.calculate_EDT', mat2str(OPT.calculate_EDT));
end

% "The sampling interval shall not be more than 1,0 mm."
if((OPT.profile_dx > 1.0 ) | (OPT.resample_to_dx > 1.0))
  error('ERROR: The sampling interval shall not be more than 1.0 mm.');
end

% "Re-sample the signal to either 0,5 mm or 1,0 mm spacing; preferably 0,5 mm."
if(~((((OPT.profile_dx     == 0.5) | (OPT.profile_dx     == 1.0)) & (OPT.resample_to_dx == 0.0)) | ...
     (((OPT.resample_to_dx == 0.5) | (OPT.resample_to_dx == 1.0)) & (OPT.profile_dx     <= 1.0))))
  % The condition above can be read as: either the texture profile should be sampled at 0.5 or 1.0 mm and no resampling,
  % OR we resample to 0.5 or 1.0 mm from a texture profile with 1.0 mm (or less) dx.
  error('ERROR: Re-sample the signal to either 0.5 mm or 1.0 mm spacing; preferably 0.5 mm.');
end

% "The length of the profile to be recorded and filtered shall be at least 1 m."
% [2021-05-17--14:14:34] Must be excluded for spot measurements.
if((OPT.spot_measurement == false) & (length(TX)*OPT.profile_dx < 1000))
  error('ERROR: The length of the profile to be recorded and filtered shall be at least 1 m.');
end

% "The minimum evaluation length over which MPD is calculated shall be 1,0 m."
if(OPT.evaluation_length < 1.0)
  error('ERROR: The minimum evaluation length over which MPD is calculated shall be 1.0 m.');
end

% Use this information to set valid MSDs below.
TX_DROPS = TX;
```

```
% Drop-out detection.  Linear interpolation and nearest neighbor extrapolation of missing values ('NaN's).
if(any(isnan(TX)))
  if(OPT.verbose)
    fprintf('7.3:  Drop-out detection.\n');
  end
  ok_samples = find(isfinite(TX));
  if(OPT.verbose)
    fprintf('7.3:  Linear interpolation and nearest neighbor extrapolation of missing values.\n');
  end
  TX = interp1(ok_samples, TX(ok_samples), [1:length(TX)], 'linear');
  % "This method of extrapolation shall be limited to a maximum length at either side of the sampled profile data series equal to 5 mm."
  % The statement above is problematic as we need to remove ALL drop-outs before filtering the profile.
  TX = interp1(ok_samples, TX(ok_samples), [1:length(TX)], 'nearest', 'extrap');
end

DX = OPT.profile_dx;
% Interpolate to 'OPT.resample_to_dx'.
if(OPT.resample_to_dx > 0.0)
  if(OPT.profile_dx > OPT.resample_to_dx)
    if(OPT.verbose)
      fprintf('7.4:  Interpolating from dx=%0.2f mm to dx=%0.2f mm.\n', OPT.profile_dx, OPT.resample_to_dx);
    end
    % Normal linear interpolation and extrapolation.
    TX = interp1(1:length(TX), TX, OPT.resample_to_dx:OPT.resample_to_dx:length(TX), 'linear', 'extrap');
  else
    if(OPT.verbose)
      fprintf('7.4:  Resampling from dx=%0.2f mm to dx=%0.2f mm.\n', OPT.profile_dx, OPT.resample_to_dx);
    end
    % Relatively fast method to perform the resampling/binning of the texture profile.
    TX = accumarray(transpose(discretize(OPT.profile_dx*[1:length(TX)], ...
                                         [0:OPT.resample_to_dx:length(TX)*OPT.profile_dx + OPT.resample_to_dx])), TX, [], @mean);
  end
  DX = OPT.resample_to_dx;
end

% Spike removal (mandatory).
if(OPT.verbose)
  fprintf('7.5:  Spike removal --- ');
end
% Spike identification.
% "The spikes are first identified in forward and reverse direction before replacing them with the interpolated value."
spike_pos_fwd = find(abs(TX(2:end) - TX(1:end - 1)) >= 3*DX) + 1;
TX = flip(TX);
spike_pos_rev = (length(TX) - 1) - find(abs(TX(2:end) - TX(1:end - 1)) >= 3*DX) + 1;
TX = flip(TX);
spike_pos = union(spike_pos_fwd, spike_pos_rev);
if(OPT.verbose)
  fprintf('%d samples affected\n', length(spike_pos));
end

% Spikes are treated as missing values.
TX(spike_pos) = NaN;

% Use this information to set valid MSDs below.
TX_SPIKE = TX;
```

```
if(length(spike_pos) > 0)
  % Linear interpolation and nearest neighbor extrapolation of the spikes.
  if(OPT.verbose)
    fprintf('7.5:  Interpolation and nearest neighbor extrapolation of the spikes.\n');
  end
  TX = interp1(find(isfinite(TX)), TX(find(isfinite(TX))), [1:length(TX)], 'linear');
  TX = interp1(find(isfinite(TX)), TX(find(isfinite(TX))), [1:length(TX)], 'nearest', 'extrap');
  TX = TX(:);
end

% Number of samples per segment: 100 for dx = 1.00; 200 for dx = 0.50.
samples_per_segment = 100*(1/DX);

% Default is continuous measurement.
if(~OPT.spot_measurement)
  % Cut the beginning mirror segment.
  if(OPT.verbose)
    fprintf('7.6:  Adding mirrored segments.\n');
  end
  beg_seg = -flip(TX(1:samples_per_segment + 1));
  beg_seg = beg_seg - 2*beg_seg(end);
  beg_seg = beg_seg(1:end - 1);

  % Cut the end mirror segment.
  end_seg = -flip(TX(end - samples_per_segment:end));
  end_seg = end_seg - 2*end_seg(1);
  end_seg = end_seg(2:end);

  % Add the mirrored segments.
  TX = [beg_seg; TX; end_seg];

  % Filter the texture profile with a high-pass Butterworth filter.
  if(OPT.verbose)
    fprintf('7.6:  Removal of long-wavelength components (high-pass filtering).\n');
  end
  [b, a] = butter(2, 1/174.2*2*DX, 'high');
  TX = filtfilt(b, a, TX);

  % Filter the texture profile with a low-pass Butterworth filter.
  if(OPT.verbose)
    fprintf('7.6:  Normalization of profile sharpness (low-pass filtering).\n');
  end
  [b, a] = butter(2, 1/2.4*2*DX, 'low');
  TX = filtfilt(b, a, TX);

  % Remove the mirrored parts.
  if(OPT.verbose)
    fprintf('7.6:  Removing mirrored segments.\n');
  end
  TX = TX(samples_per_segment + 1:end - (samples_per_segment));
end

% Reshape the texture profile to a matrix with one segment per row.
TX_MAT = transpose(reshape(TX(1:samples_per_segment*floor(length(TX)/samples_per_segment)), ...
                           samples_per_segment, floor(length(TX)/samples_per_segment)));
```

```
if(OPT.spot_measurement)
  % Remove the mean value from all segments.
  y_star = transpose(bsxfun(@minus, transpose(TX_MAT), mean(transpose(TX_MAT))));

  % The slope of each profile segment.
  BETA = (y_star*transpose([-100/2 + DX/2:DX:100/2 - DX/2]))/sum([-100/2 + DX/2:DX:100/2 - DX/2].^2);

  % The intercept of each profile segment.
  ALPHA = mean(TX_MAT, 2) - (100/2 + DX/2)*BETA;

  % Slope suppression on each profile segment.
  TX_MAT = TX_MAT - bsxfun(@plus, bsxfun(@times, repmat(DX*[1:samples_per_segment], length(BETA), 1), BETA), ALPHA);
end

% Calculate the MSDs.
if(OPT.verbose)
  fprintf('7.8:  Calculate the MSDs.\n');
end

% Peak on first segment (A).
PSA_max = max(TX_MAT(:, 1:samples_per_segment/2), [], 2);

% Peak on second segment (B).
PSB_max = max(TX_MAT(:, samples_per_segment/2 + 1:samples_per_segment), [], 2);

% Mean of the entire segment.
PS_mean = nanmean(TX_MAT, 2);

% The MSD is "The MSD for each individual segment is the arithmetic mean of the two peak levels minus the average profile level".
RES.MSD = transpose((PSA_max + PSB_max)/2 - PS_mean);

% "For the segment to be valid, the removed spikes should not correspond to more than 5 % of the total segment profile (after resampling)."
TX_SPIKE_MAT = transpose(reshape(TX_SPIKE(1:samples_per_segment*floor(length(TX_SPIKE)/samples_per_segment)), ...
                                 samples_per_segment, floor(length(TX_SPIKE)/samples_per_segment)));
spike_hit = find(sum(isnan(TX_SPIKE_MAT), 2) > 0.05*samples_per_segment);
if(OPT.verbose)
  fprintf('7.5:  Set MSDs with more than 5 %% spikes to invalid (%d segments affected).\n', length(spike_hit));
end
RES.MSD(spike_hit) = NaN;

% "Segments with loss of data due to drop-out greater than 10 % (of the total number of samples) shall be discarded." [The "quote" is edited.]
if((OPT.profile_dx ~= 0.5) & (OPT.profile_dx ~= 1.0))
  % Find drop-outs at a non 0.5 or 1.0 sample distance.
  drops_hit = find(100*(accumarray(transpose(discretize(OPT.profile_dx*[1:length(TX_DROPS)], [0:100:2*(length(TX_DROPS)*OPT.profile_dx)])), ...
                                   isnan(TX_DROPS), [], @sum)./ ...
                        accumarray(transpose(discretize(OPT.profile_dx*[1:length(TX_DROPS)], [0:100:2*(length(TX_DROPS)*OPT.profile_dx)])), ...
                                   ones(size(TX_DROPS)), [], @sum)) > 10);
else
  % Find drop-outs at a 0.5 or 1.0 sample distance.
  TX_DROPS_MAT = transpose(reshape(TX_DROPS(1:samples_per_segment*floor(length(TX_DROPS)/samples_per_segment)), ...
                                   samples_per_segment, floor(length(TX_DROPS)/samples_per_segment)));
  drops_hit = find(sum(isnan(TX_DROPS_MAT), 2) > 0.10*samples_per_segment);
end
if(OPT.verbose)
  fprintf('7.5:  Set MSDs with more than 10 %% drop-outs to invalid (%d segments affected).\n', length(drops_hit));
```

```
end
RES.MSD(drops_hit) = NaN;

% Check drop-outs at the beginning or end of the profile.  "This method of extrapolation shall be limited to a maximum length at either
% side of the sampled profile data series equal to 5 mm."
% Check the first segment.
if(DX*(min(find(isfinite(TX_DROPS)))-1) > 5)
  RES.MSD(001) = NaN;
end
% Check the last segment.
if(DX*(length(TX_DROPS) - max(find(isfinite(TX_DROPS)))) > 5)
  RES.MSD(end) = NaN;
end

% Extreme MSD value removal (optional).
if(OPT.extreme_MSD_remove)
  if(OPT.verbose)
    fprintf('7.9:  Extreme MSD value removal.\n');
  end
  RES.MSD = movmedian(RES.MSD, 3);
end

% Number of segments per evaluation length.
segms_per_pres_len = floor((1000*OPT.evaluation_length)/100);

% Number of whole units of 'segms_per_pres_len'.
DIST = floor(length(TX)/(samples_per_segment*segms_per_pres_len));

if(OPT.verbose)
  fprintf('7.10: Averaging of MSD to determine the MPD.\n');
end
% Reshape the MSD-vector to have 'segms_per_pres_len' values per row, and 'DIST' rows.  Also, calculate the 'MPD' per presentation length (excluding 'NaN':s).
RES.MPD = nanmean(reshape(RES.MSD(1:segms_per_pres_len*DIST), segms_per_pres_len, DIST));

% The standard deviation of the MSD values per evaluation length.
RES.STD = nanstd(reshape(RES.MSD(1:segms_per_pres_len*DIST), segms_per_pres_len, DIST));

% Find evaluation length sections with more than 50% invalid MSDs.
MPD_INV = find(100*(sum(isnan(reshape(RES.MSD(1:segms_per_pres_len*DIST), segms_per_pres_len, DIST)))./segms_per_pres_len) > 50);
if(~isempty(MPD_INV))
  if(OPT.verbose)
    fprintf('7.**: Setting MPDs with too few MSDs to invalid.\n');
  end
  RES.MPD(MPD_INV) = NaN;
  RES.STD(MPD_INV) = NaN;
end

% Calculate the ETD.
if(OPT.calculate_EDT)
  if(OPT.verbose)
    fprintf('7.11: Calculation of ETD.\n');
  end
  RES.EDT = 1.1*RES.MPD;
end
```