```
function [ handles ] = Compute_SPL_by_Vehicle_Side_forGUI( handles )
% This function computes sound pressure level for each microphone (driver,
% passenger, front, and trigger)

% This function is called separately for each operating condition in the
% following order:  {'Reverse'}    {'Stationary'}    {'10 km/h'}    {'20 km/h'}    {'30 km/h'}

numSamplesLengthForLoop = length(unique( handles.data( handles.data.opCond == handles.opCondForLoop, { 'sampleID' } ) ) );

% Pull all samples for current operating condition only
driverDataList = ( handles.data( handles.data.opCond == handles.opCondForLoop & handles.data.micSide == 'driver' & handles.data.signalType == 'signal', :) );
preAmbDriverDataList = ( handles.data( handles.data.opCond == handles.opCondForLoop & handles.data.micSide == 'driver' & handles.data.signalType == 'preAmbient', :) );
postAmbDriverDataList = ( handles.data( handles.data.opCond == handles.opCondForLoop & handles.data.micSide == 'driver' & handles.data.signalType == 'postAmbient', :) );

passengerDataList = ( handles.data( handles.data.opCond == handles.opCondForLoop & handles.data.micSide == 'passenger' & handles.data.signalType == 'signal', :) );
preAmbPassengerDataList = ( handles.data( handles.data.opCond == handles.opCondForLoop & handles.data.micSide == 'passenger' & handles.data.signalType == 'preAmbient', :) );
postAmbPassengerDataList = ( handles.data( handles.data.opCond == handles.opCondForLoop & handles.data.micSide == 'passenger' & handles.data.signalType == 'postAmbient', :) );

frontDataList = ( handles.data( handles.data.opCond == handles.opCondForLoop & handles.data.micSide == 'front' & handles.data.signalType == 'signal', :) );
preAmbFrontDataList = ( handles.data( handles.data.opCond == handles.opCondForLoop & handles.data.micSide == 'front' & handles.data.signalType == 'preAmbient', :) );
postAmbFrontDataList = ( handles.data( handles.data.opCond == handles.opCondForLoop & handles.data.micSide == 'front' & handles.data.signalType == 'postAmbient', :) );

triggerDataList = ( handles.data( handles.data.opCond == handles.opCondForLoop & handles.data.micSide == 'all' & handles.data.signalType == 'trigger', :) );

%% Display current operating condition and total operations for current operating condition
set( handles.textProcessingStep, 'String', 'Processing sample' );
set( handles.displayRunStatusCurrentFile, 'Visible', 'on' );
set( handles.textRunStatusOf, 'Visible', 'on');
set( handles.displayRunStatusTotalFiles, 'Visible', 'on' );
set( handles.textRunStatusFor, 'Visible', 'on');
set( handles.displayCurrentOC, 'Visible', 'on');

totalSamplesforOC = num2str( length( driverDataList ) );
set( handles.displayRunStatusTotalFiles, 'String', totalSamplesforOC );
set( handles.displayCurrentOC, 'String', handles.opCondForLoop );
drawnow

%% Prep GUI inputs and run through SLM for current OC for all samples

```

```
for cnt = 1 : numSamplesLengthForLoop

    % ----------------------------------------------------------------
    % For stop button
    drawnow
    if get(handles.pushbuttonStop, 'userdata') % stop condition
        handles.stopButtonFlag = 'true' ;
        return;
    end
    % ----------------------------------------------------------------

    set( handles.displayRunStatusCurrentFile, 'String', num2str( cnt ) );
    drawnow

    fprintf( '\n\nOperating Condition: %s, Sample: %d\n', handles.opCondForLoop, cnt )

    % Prep GUI inputs for input into SLM
    % This will load wav files and resample if needed

    % Driver signal, pre, and post ambient
    driver.inputFileName = char( driverDataList.fileName( cnt ) );
    driver.inputPathName = char( driverDataList.pathName( cnt ) );
    [driverData, fs] = wavread( [  driver.inputPathName driver.inputFileName ] ) ;
    driverData = driverData( : , driverDataList.channel( cnt ) );
    if fs ~= 44100 % Resample if needed
        driverData = resample( driverData, 44100, fs ); %fs is not used after this, do not need to reassign to 44100
        fs = 44100; % Not needed currently, but included in case new code is added that uses this
    end

    preAmbDriver.inputFileName = char( preAmbDriverDataList.fileName( cnt ) );
    preAmbDriver.inputPathName = char( preAmbDriverDataList.pathName( cnt ) );
    [preAmbDriverData, fs] = wavread( [  preAmbDriver.inputPathName preAmbDriver.inputFileName ] ) ;
    preAmbDriverData = preAmbDriverData( : , preAmbDriverDataList.channel( cnt ) );
    if fs ~= 44100 % Resample if needed
        preAmbDriverData = resample( preAmbDriverData, 44100, fs );
        fs = 44100; % Not needed currently, but included in case new code is added that uses this
    end

    postAmbDriver.inputFileName = char( postAmbDriverDataList.fileName( cnt ) );
    postAmbDriver.inputPathName = char( postAmbDriverDataList.pathName( cnt ) );
    [postAmbDriverData, fs] = wavread( [  postAmbDriver.inputPathName postAmbDriver.inputFileName ] ) ;
    postAmbDriverData = postAmbDriverData( : , postAmbDriverDataList.channel( cnt ) );
    if fs ~= 44100 % Resample if needed
        postAmbDriverData = resample( postAmbDriverData, 44100, fs );
        fs = 44100;
    end

    % Passenger signal, pre, and post ambient
    passenger.inputFileName = char( passengerDataList.fileName( cnt ) );
    passenger.inputPathName = char( passengerDataList.pathName( cnt ) );
```

```
    [passengerData, fs] = wavread( [  passenger.inputPathName passenger.inputFileName ] ) ;
    passengerData = passengerData( : , passengerDataList.channel( cnt ) );
    if fs ~= 44100 % Resample if needed
        passengerData = resample( passengerData, 44100, fs );
        fs = 44100; % Not needed currently, but included in case new code is added that uses this
    end

    preAmbPassenger.inputFileName = char( preAmbPassengerDataList.fileName( cnt ) );
    preAmbPassenger.inputPathName = char( preAmbPassengerDataList.pathName( cnt ) );
    [preAmbPassengerData, fs] = wavread( [  preAmbPassenger.inputPathName preAmbPassenger.inputFileName ] ) ;
    preAmbPassengerData = preAmbPassengerData( : , preAmbPassengerDataList.channel( cnt ) );
    if fs ~= 44100 % Resample if needed
        preAmbPassengerData = resample( preAmbPassengerData, 44100, fs );
        fs = 44100; % Not needed currently, but included in case new code is added that uses this
    end

    postAmbPassenger.inputFileName = char( postAmbPassengerDataList.fileName( cnt ) );
    postAmbPassenger.inputPathName = char( postAmbPassengerDataList.pathName( cnt ) );
    [postAmbPassengerData, fs] = wavread( [  postAmbPassenger.inputPathName postAmbPassenger.inputFileName ] ) ;
    postAmbPassengerData = postAmbPassengerData( : , postAmbPassengerDataList.channel( cnt ) );
    if fs ~= 44100 % Resample if needed
        postAmbPassengerData = resample( postAmbPassengerData, 44100, fs );
        fs = 44100; % Not needed currently, but included in case new code is added that uses this
    end

    if strcmp(handles.opCondForLoop, 'Stationary')
    % These statements should be called the second time the function is
    % called (the first time it is called is for Reverse, later calls are
    % for passbys)
        front.inputFileName = char( frontDataList.fileName( cnt ) );
        front.inputPathName = char( frontDataList.pathName( cnt ) );
        [frontData, fs] = wavread( [  front.inputPathName front.inputFileName ] ) ;
        frontData = frontData( : , frontDataList.channel( cnt ) );
        if fs ~= 44100 % Resample if needed
            frontData = resample( frontData, 44100, fs );
            fs = 44100; % Not needed currently, but included in case new code is added that uses this
        end

        preAmbFront.inputFileName = char( preAmbFrontDataList.fileName( cnt ) );
        preAmbFront.inputPathName = char( preAmbFrontDataList.pathName( cnt ) );
        [preAmbFrontData, fs] = wavread( [  preAmbFront.inputPathName preAmbFront.inputFileName ] ) ;
        preAmbFrontData = preAmbFrontData( : , preAmbFrontDataList.channel( cnt ) );
        if fs ~= 44100 % Resample if needed
            preAmbFrontData = resample( preAmbFrontData, 44100, fs );
            fs = 44100; % Not needed currently, but included in case new code is added that uses this
        end

        postAmbFront.inputFileName = char( postAmbFrontDataList.fileName( cnt ) );
        postAmbFront.inputPathName = char( postAmbFrontDataList.pathName( cnt ) );
        [postAmbFrontData, fs] = wavread( [  postAmbFront.inputPathName postAmbFront.inputFileName ] ) ;
```

```
140         postAmbFrontData = postAmbFrontData( : , postAmbFrontDataList.channel( cnt ) );
141         if fs ~= 44100 % Resample if needed
142             postAmbFrontData = resample( postAmbFrontData, 44100, fs );
143             fs = 44100; % Not needed currently, but included in case new code is added that uses this
144         end
145
146     end
147
148     %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
149
150     % Determine indeces associated with triggers
151     % Note: Ambients do not need to have idxStart / Stop determined, their
152     % start / stop is determined in the Min_SLM function
153
154     % ----------------------------------------------------------------
155     % For stop button
156     drawnow
157     if get(handles.pushbuttonStop, 'userdata') % stop condition
158         handles.stopButtonFlag = 'true' ;
159         return;
160     end
161     % ----------------------------------------------------------------
162
163     % For passbys, trim driver/passenger signal data at trigger times
164     % Code assumes two triggers, one for AA' and one for PP'
165     if strcmp(handles.opCondForLoop,'10 km/h' ) || strcmp (handles.opCondForLoop,'20 km/h' )||  strcmp
        (handles.opCondForLoop,'30 km/h' );
166
167         % Get location of trigger for this sample of the given Op Cond
168         trigger.inputFileName = char( triggerDataList.fileName( cnt ) );
169         trigger.inputPathName = char( triggerDataList.pathName( cnt ) );
170
171         % TriggerThreshold column of the triggerDataList (comes from handles.data)
172         thresholdPercent = triggerDataList.TriggerThreshold{ cnt }; % Should be a double here
173
174         [ idxStart, idxStop ] = getPassbyTrimRegion( trigger.inputFileName, trigger.inputPathName, thresholdPercent,
            triggerDataList.channel( cnt ) );
175
176         % Same trigger is used for driver and passenger sides
177         idxStartDriver = idxStart;
178         idxStartPassenger = idxStart;
179         idxStopDriver = idxStop;
180         idxStopPassenger = idxStop;
181
182     end
183
184     % Trim the Stationary signal to 10 sec (10 * fs) per final rule section 7.1.1(c)
185     % Method envoked keeps the last part of the measurement and trims off
186     % the first part (where filters may still be stabilizing)
187     if strcmp( handles.opCondForLoop, 'Stationary' )
```

```
        [ idxStartDriver, idxStopDriver ] = getStationaryTrimRegion( driverData, driver.inputFileName );
        [ idxStartPassenger, idxStopPassenger ] = getStationaryTrimRegion( passengerData, passenger.inputFileName );
        [ idxStartFront, idxStopFront ] = getStationaryTrimRegion( frontData, front.inputFileName );
    end

    % Set Analysis Region (Trim) for the Reverse signal to 10 sec (10 * fs) per final rule section 7.1.1(c)
    % Method envoked keeps the last part of the measurement and trims off
    % the first part (where filters may still be stabilizing)
    if strcmp( handles.opCondForLoop, 'Reverse' )
        [ idxStartDriver, idxStopDriver ] = getStationaryTrimRegion( driverData, driver.inputFileName );
        [ idxStartPassenger, idxStopPassenger ] = getStationaryTrimRegion( passengerData, passenger.inputFileName );
    end

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    % SPL COMPUTATIONS

    % Process Vehicle Signal for Driver Side
    disp( '     Computing SPL for Vehicle - Driver Side.' )
    [ driverSlmResults ] = Max_SLM_A_Fast_44p1_kHz( driverData, idxStartDriver, idxStopDriver );
    printDebugLog( driverSlmResults, idxStartDriver, idxStopDriver, driver.inputFileName, driver.inputPathName
    ,'driver', handles );
    driver.vehicleOverallLevels_dBA( cnt )  = driverSlmResults.maxSPL;
    driver.vehicleFOBLevels_dBA( cnt, : ) = driverSlmResults.oneThirdOctaveBandLevelsAtMax;
    driver.inputFileNames( cnt ) = { driver.inputFileName } ;

    % Process Pre-Ambient Signal for Driver Side
    disp( '     Computing SPL for Pre-Ambient - Driver Side.' )
    [ preAmbDriverSlmResults] = Min_SLM_A_Fast_44p1_kHz( preAmbDriverData );
    printDebugLog( preAmbDriverSlmResults, 44100, length( preAmbDriverData ), preAmbDriver.inputFileName,
    preAmbDriver.inputPathName ,'pre-ambient driver', handles);

    % Process Post-Ambient Signal for Driver Side
    disp( '     Computing SPL for Post-Ambient - Driver Side.' )
    [ postAmbDriverslmResults ] = Min_SLM_A_Fast_44p1_kHz( postAmbDriverData ); % Note: Min_SLM function has built in
    index determination
    printDebugLog( postAmbDriverslmResults, 44100, length( postAmbDriverData ), postAmbDriver.inputFileName,
    postAmbDriver.inputPathName ,'post-ambient driver', handles);

    %--- Pick the ambient measurement with the lower minimum level
    if preAmbDriverSlmResults.minSPL < postAmbDriverslmResults.minSPL
        driver.ambientOverallLevels_dBA( cnt ) = preAmbDriverSlmResults.minSPL;
        driver.ambientFOBLevels_dBA( cnt, : ) = preAmbDriverSlmResults.oneThirdOctaveBandLevelsAtMin;
        driver.ambientFileNames( cnt ) = { preAmbDriver.inputFileName };
        driver.ambientLowerSide( cnt ) = { '(Pre-Ambient)' };

    else
        driver.ambientOverallLevels_dBA( cnt ) = postAmbDriverslmResults.minSPL;
        driver.ambientFOBLevels_dBA( cnt, : ) = postAmbDriverslmResults.oneThirdOctaveBandLevelsAtMin;
        driver.ambientFileNames( cnt ) = { postAmbDriver.inputFileName } ;
        driver.ambientLowerSide( cnt ) = { '(Post-Ambient)' };
```

```
    end

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

    % Process Vehicle Signal for Passenger Side
    disp( '     Computing SPL for Vehicle - Passenger Side.' )
    [ passengerSlmResults ] = Max_SLM_A_Fast_44p1_kHz( passengerData, idxStartPassenger, idxStopPassenger );
    printDebugLog( passengerSlmResults, idxStartPassenger, idxStopPassenger, passenger.inputFileName, passenger.inputPathName ,'passenger', handles);
    passenger.vehicleOverallLevels_dBA( cnt ) = passengerSlmResults.maxSPL;
    passenger.vehicleFOBLevels_dBA( cnt, : ) = passengerSlmResults.oneThirdOctaveBandLevelsAtMax;
    passenger.inputFileNames( cnt ) = { passenger.inputFileName  } ;

    % Process Pre-Ambient Signal for Passenger Side
    disp( '     Computing SPL for Pre-Ambient - Passenger Side.' )
    [ preAmbPassengerSlmResults ] = Min_SLM_A_Fast_44p1_kHz( preAmbPassengerData );
    printDebugLog( preAmbPassengerSlmResults, 44100, length( preAmbPassengerData ), preAmbPassenger.inputFileName, preAmbPassenger.inputPathName ,'pre-ambient passenger', handles);

    % Process Post-Ambient Signal for Passenger Side
    disp( '     Computing SPL for Post-Ambient - Passenger Side.' )
    [ postAmbPassengerslmResultsPost ] = Min_SLM_A_Fast_44p1_kHz( postAmbPassengerData );
    printDebugLog( postAmbPassengerslmResultsPost, 44100, length( postAmbPassengerData ), postAmbPassenger.inputFileName, postAmbPassenger.inputPathName ,'post-ambient passenger', handles);

    %--- Pick the ambient measurement with the lower minimum level
    if preAmbPassengerSlmResults.minSPL < postAmbPassengerslmResultsPost.minSPL
        passenger.ambientOverallLevels_dBA( cnt ) = preAmbPassengerSlmResults.minSPL;
        passenger.ambientFOBLevels_dBA( cnt, : ) = preAmbPassengerSlmResults.oneThirdOctaveBandLevelsAtMin;
        passenger.ambientFileNames( cnt ) = { preAmbPassenger.inputFileName };
        passenger.ambientLowerSide( cnt ) = { '(Pre-Ambient)' };

    else
        passenger.ambientOverallLevels_dBA( cnt ) = postAmbPassengerslmResultsPost.minSPL;
        passenger.ambientFOBLevels_dBA( cnt, : ) = postAmbPassengerslmResultsPost.oneThirdOctaveBandLevelsAtMin;
        passenger.ambientFileNames( cnt ) = { postAmbPassenger.inputFileName } ;
        passenger.ambientLowerSide( cnt ) = { '(Post-Ambient)' };
    end

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    % Select Stationary "Front" Files

    if strcmp(handles.opCondForLoop, 'Stationary')
        disp( '     Computing SPL for Vehicle - Front Side.' )
        [ frontSlmResults ] = Max_SLM_A_Fast_44p1_kHz( frontData, idxStartFront, idxStopFront );
        printDebugLog( frontSlmResults, idxStartFront, idxStopFront, front.inputFileName, front.inputPathName ,'front', handles);
        front.vehicleOverallLevels_dBA( cnt ) = frontSlmResults.maxSPL;
        front.vehicleFOBLevels_dBA( cnt, : ) = frontSlmResults.oneThirdOctaveBandLevelsAtMax;
        front.inputFileNames( cnt ) = { front.inputFileName  };
```

```
280
281         % Process Pre-Ambient Signal for Front Side
282         disp( '     Computing SPL for Pre-Ambient - Front Side.' )
283         [ preAmbFrontSlmResults] = Min_SLM_A_Fast_44p1_kHz( preAmbFrontData );
284         printDebugLog( preAmbFrontSlmResults, 44100, length( preAmbFrontData ), preAmbFront.inputFileName,
            preAmbFront.inputPathName ,'pre-ambient front', handles);
285
286         % Process Ambient Signals for Front Side
287         disp( '     Computing SPL for Post-Ambient.' )
288         [ postAmbFrontSlmResults ] = Min_SLM_A_Fast_44p1_kHz( postAmbFrontData );
289         printDebugLog( postAmbFrontSlmResults, 44100, length( postAmbFrontData ), postAmbFront.inputFileName,
            postAmbFront.inputPathName ,'post-ambient front', handles);
290
291         %--- Pick the ambient measurement with the lower minimum level
292         if preAmbFrontSlmResults.minSPL < postAmbFrontSlmResults.minSPL
293             front.ambientOverallLevels_dBA( cnt ) = preAmbFrontSlmResults.minSPL;
294             front.ambientFOBLevels_dBA( cnt, : ) = preAmbFrontSlmResults.oneThirdOctaveBandLevelsAtMin;
295             front.ambientFileNames( cnt ) = { preAmbFront.inputFileName };
296             front.ambientLowerSide( cnt ) = { '(Pre-Ambient)' };
297
298         else
299             front.ambientOverallLevels_dBA( cnt ) = postAmbFrontSlmResults.minSPL;
300             front.ambientFOBLevels_dBA( cnt, : ) = postAmbFrontSlmResults.oneThirdOctaveBandLevelsAtMin;
301             front.ambientFileNames( cnt ) = { postAmbFront.inputFileName } ;
302             front.ambientLowerSide( cnt ) = { '(Post-Ambient)' };
303         end
304
305     else
306         front.vehicleOverallLevels_dBA( cnt ) = -99;
307         front.vehicleFOBLevels_dBA( cnt, : ) = -99 * ones( 29, 1 );
308         front.ambientOverallLevels_dBA( cnt ) = -99;
309         front.ambientFOBLevels_dBA( cnt, : ) = -99 * ones( 29, 1 );
310         front.inputFileNames( cnt ) = { [] };
311         front.ambientLowerSide( cnt ) = { [] };
312
313         handles.frontSPL = front ;
314     end
315
316     handles.driverSPL = driver ;
317     handles.passengerSPL = passenger ;
318     handles.frontSPL = front ;
319
320 end
```