```
%{
This file takes in the struct containing the filtered data and extract
 all
the time-domain features:
    1. Maximum
    2. Minimum
    3. Mean
    4. Standard deviation
    5. Root means square (RMS)
    6. Zero crossing
    7. Maximum slope changes

    The notations for them: "MAX", "MIN", "AVG", "SD", "RMS", "ZC",
 "MSC".

Arguments:
- `filteredRawData`   -> the struct with the raw data after going
 through
                            the low pass filter.

Returns:
- `processedData`  -> struct with the time-domain features extracted
                        according to the given time window and
 interval
%}

function processedData = extractFeatures(filteredRawData)
    % ----------------------------------------------
    % Loop through all of the filteredRawData, extract max, min, mean,
 standard
    % deviation and RMS.
    fprintf("\nExtracting time domain features...\n")
    % ----------------------------------------------
    % array containing the names of the activities.
    % These names will match the field names in the struct
    sets = ["LGW","RA","RD","SiS","StS"];
    % array containing the names of the time domain features.
    % These names will match the field names in the struct.
    features = ["MAX", "MIN", "AVG", "SD","RMS"];
    % size of the sliding window for extracting features in
 milliseconds
    window_duration = 400;
    % define the time interval required in ms
    timeInterval = 50;
    % loop through each of the folders
    for ff = 1 : length(sets)
        for kk = 1 : length(filteredRawData)
            current_dataset = table2array(filteredRawData(kk).
(sets{ff}));
            current_dataset_without_timestamp =
 current_dataset(:,2:end);
            current_timestamp = current_dataset(:,1);
```

```
            % -------------------------------------------------------
            % We should not assume that the timestep between samples
is fixed.
            % Therefore, we find the average timestep for every
dataset and
            % change the window size accordingly.

            % find the average timestep between rows
            avg_timestep = mean(diff(current_timestamp));
            % in case the timestep is in seconds rather than
milliseconds
            if avg_timestep < 1
                avg_timestep = avg_timestep*1000;
            end
            % `window_size` defines the number of readings in each
window
            window_size = int32(window_duration/avg_timestep);
            %-------------------------------------------------------

            % extract time domain features from each dataset,
discarding
            % endpoints to have all windows exactly the length they
should be.
            smean = movmean(current_dataset_without_timestamp,
window_size, 'Endpoints','discard');
            stdD = movstd(current_dataset_without_timestamp,
window_size, 'Endpoints','discard');
            maxV = movmax(current_dataset_without_timestamp,
window_size, 'Endpoints','discard');
            min = movmin(current_dataset_without_timestamp,
window_size, 'Endpoints','discard');
            rms = sqrt(movmean(current_dataset_without_timestamp .^ 2,
window_size, 'Endpoints','discard'));
            % group all the features
            dataset_features = {maxV, min, smean, stdD, rms};
            % assign all the features to a single new struct
            for w = 1 : length(dataset_features)
                % reduce the data using the required time interval.
                % We want to extract only every Nth row to match our
time
                % interval. The 'interval' var defines N.
                [reduced_data, interval] =
reduceData(current_timestamp, dataset_features{1,w}, timeInterval);
                temp_processedData(1).(features{w}) = reduced_data;
            end
            processedData(kk).(sets{ff}) = temp_processedData;
        end
    end

    % ----------------------------------------------
    % Loop through all of the filteredRawData, extract Zero Crossing.
    fprintf("\nManually extracting Zero Crossing...\n")
    % ----------------------------------------------
```

```
   % extract zero crossing for the data and add to the same
processedData
   % struct
   % loop through each of the folders
   for ff = 1 : length(sets)
       for kk = 1 : length(filteredRawData)
           % fprintf("\nIn set number %i and dataset number %i, in
set %s\n", ff, kk, sets(ff))
           current_dataset = filteredRawData(kk).(sets{ff});
           current_dataset_without_timestamp =
current_dataset(:,2:end);
           current_timestamp = table2array(current_dataset(:,1));
           % -------------------------------------------------------
           % We should not assume that the timestep between samples
is fixed.
           % Therefore, we find the average timestep for every
dataset and
           % change the window size accordingly.

           % find the average timestep between rows
           avg_timestep = mean(diff(current_timestamp));
           % in case the timestep is in seconds rather than
milliseconds
           if avg_timestep < 1
               avg_timestep = avg_timestep*1000;
           end
           % `window_size` defines the number of readings in each
window
           window_size = int16(window_duration/avg_timestep);
           % define the half window size according to whether the
window is
           % odd or even
           if rem(window_size,2) == 0
               % if window size is even
               half_window_size = int16(window_size/2);
           else
               % if window size is odd
               half_window_size = int16((window_size-1)/2);
           end
           %-------------------------------------------------------
           % Filter the data
           % loop through the columns in the single dataset
           clearvars zc_dataset
           for ii = 1 : width(current_dataset_without_timestamp)
               % obtain the relevant column
               colm =
table2array(current_dataset_without_timestamp(1:end,ii));
               % calculate zero crossing manually
               % (for each window, 1 if a ZC exists, 0 if not)
               zc = false;
               clearvars zc_column
               % loop through the column
               for r = 1 : interval :length(colm)
                   if length(colm) < (window_size+2)
```

```
                        % if the column is smaller than the window
 size then ignore
                        fprintf("\nHasal?\n")
                        continue
                    elseif r > (length(colm)-(half_window_size))
                        % if towards the end of the column, ignore the
 window
                        continue
                    elseif r < (half_window_size+1)
                        % if towards the start of the column, ignore
 the window
                        continue
                    else
                        if rem(window_size,2) == 0
                            % if window size is even
                            g = (window_size-2)/2;
                            h = colm(r-(g+1):r+g);
                        else
                            % if window size is odd
                            h = colm(r-half_window_size:r
+half_window_size);
                        end
                    end
                    % loop through h and see if a ZC exists
                    for rr = 1 : length(h)-1
                        if (h(rr)*h(rr+1))<0
                            zc = true;
                        end
                    end
                    % if we had found a ZC then we want to assign 1,
 if not
                    % then 0. Indexing `sequentialIndex` instead of r
 as r is
                    % increasing by non-1 increments.
                    sequentialIndex = ((r-1)/interval)-3;
                    if zc
                        zc_column(sequentialIndex) = 1;
                    else
                        zc_column(sequentialIndex) = 0;
                    end
                    zc = false;
                end
                % append the zc_column to the existing zc table
                zc_dataset(:,ii) = zc_column;
            end
            processedData(kk).(sets{ff})(1).ZC = zc_dataset;
        end
    end



    % ---------------------------------------------
    % Loop through all of the filteredRawData, extract Maximum Slope
 Change.
```

```
    fprintf("\nManually extracting MSC...\n")
    % ---------------------------------------------
    % extract zero crossing for the data and add to the same
processedData
    % struct
    % loop through each of the folders
    for ff = 1 : length(sets)
        for kk = 1 : length(filteredRawData)
            current_dataset = filteredRawData(kk).(sets{ff});
            current_dataset_without_timestamp =
current_dataset(:,2:end);
            current_timestamp = table2array(current_dataset(:,1));
            % -------------------------------------------------------
            % We should not assume that the timestep between samples
is fixed.
            % Therefore, we find the average timestep for every
dataset and
            % change the window size accordingly.

            % find the average timestep between rows
            avg_timestep = mean(diff(current_timestamp));
            % in case the timestep is in seconds rather than
milliseconds
            if avg_timestep < 1
                avg_timestep = avg_timestep*1000;
            end
            % `window_size` defines the number of readings in each
window
            window_size = int16(window_duration/avg_timestep);
            % define the half window size according to whether the
window is
            % odd or even
            if rem(window_size,2) == 0
                % if window size is even
                half_window_size = int16(window_size/2);
            else
                % if window size is odd
                half_window_size = int16((window_size-1)/2);
            end
            %-------------------------------------------------------
            % Filter the data
            % loop through the columns in the single dataset
            clearvars ms_dataset
            for ii = 1 : width(current_dataset_without_timestamp)
                % obtain the relevant column
                colm =
table2array(current_dataset_without_timestamp(1:end,ii));
                clearvars ms_column
                % loop through the column
                for r = 1 : interval :length(colm)
                    if length(colm) < (window_size+2)
                        % if the column is smaller than the window
size then ignore
                        continue
```

```
                    elseif r > (length(colm)-(half_window_size))
                        % if towards the end of the column, ignore the
 window
                        continue
                    elseif r < (half_window_size+1)
                        % if towards the start of the column, ignore
 the window
                        continue
                    else
                        % under normal conditions, take window/2
 values before
                        % r and window/2 values after it
                        if rem(window_size,2) == 0
                            g = (window_size-2)/2;
                            % if window size is evn
                            h = colm(r-(g+1):r+g);
                            timeColum = current_timestamp(r-(g+1):r
+g);
                        else
                            % if window size is odd
                            h = colm(r-half_window_size:r
+half_window_size);
                            timeColum = current_timestamp(r-
half_window_size:r+half_window_size);
                        end
                    end
                    sequentialIndex = ((r-1)/interval)-3;
                    % gradient() finds the slope change between
 consequetive
                    % data points.
                    dydx = gradient(h) ./ gradient(timeColum);
                    % we now need to find the differences between
 consequtive
                    % gradients, then find the maximum value i.e. max
 slope
                    % change
                    maxSlopeChange = max(diff(dydx));
                    ms_column(sequentialIndex) = maxSlopeChange;
                end
                % append the ms_column to the existing ms table
                ms_dataset(:,ii) = ms_column;
            end
            processedData(kk).(sets{ff})(1).MSC = ms_dataset;
        end
    end
end
```

Published with MATLAB® R2020a