Perception Tools Dev Log

Pengfei LIU

1 MDF Signal Conversion

Before using the Perception tool, it is necessary to convert the raw signals recorded in MF4 format by CANape into .mat format, which can be directly used by MATLAB. This conversion is achieved through the `convertLog.m` script, which utilizes the `Read_MDF_DM.m`, `readMDF_DM_MD.m`, and Toolbox functions.

For the SWEET 200 signals, there are no issues. When converting SWEET 420 signals using this script, there are cases where signal names longer than 63 characters are truncated. This issue is from MATLAB's limitation on variable name lengths, with some SWEET 420 signal names being quite long.

To address this issue, I referred to the algorithm in `Read_MDF_DM_AC.m` to shorten the signal names. The approach is as follows:

- Checking if the signal name contains a dot (.):
  - If the signal name contains a dot, it is split into different parts `(strsplit(curNames{i}{j}, '.'))`. Any possible square brackets in each part are removed before recombining them. Typically, the last two parts are combined, especially if their combined length is less than 62 characters. If this combined length is still too long, only the last part is used as the new signal name.
- Handling brackets:
  - Replace left brackets `[` with an underscore `_`.
  - Remove right brackets `]`.
- Handling non-dotted and overly long names:
  - If the signal name does not contain a dot and exceeds 63 characters, the last 63 characters of the signal name are directly used.

The algorithm in `Read_MDF_DM_AC.m`:

```
for i=1:length(curNames(:,1))
    timeVectors{i, 1} = ['t' num2str(i)];
    try timeVectors{i, 2} = read(mdfObj, i, mdfObj.ChannelNames{i}{curSizes(i)},
'OutputFormat','Vector');
        if isempty(timeVectors{i,2})
            timeVectorsErrors(i, 1) = 1;
        end
    catch
        timeVectorsErrors(i, 1) = 1;
    end
    for j=1:(curSizes(i) - 1)
        % ici on check chaque signal du groupe de signaux :

        if length(curNames{i}{j})>63 %Si le nom est trop fat, on enlève les préfix
            if contains(curNames{i}{j},'.')
                temp = strsplit(curNames{i}{j},'.');
                temp2 = string(temp);
                temp2 = strrep(strrep(temp2,']',''),'[','_');

                if (length(char(temp2(:,end))) + length(char(temp2(:,end-1))))<62
                    curNames{i}{j} = [char(temp2(:,end-1)) '_' char(temp2(:,end))];
                else
                    curNames{i}{j} = char(temp(:,end));
                end
            else % si il n'y a pas de préfix, on guillotine, gardant la fin pour distinguer chaque
signaux correctement
                curNames{i}{j}(length(curNames{i}{j})-63:end);
            end
        end
        if contains(curNames{i}{j},'[')
            curNames{i}{j} = strrep(curNames{i}{j},'[','_');

        end
        if contains(curNames{i}{j},']')
                       curNames{i}{j} = strrep(curNames{i}{j},']','');
        end
            if isempty(listVars) || any(contains(listVars,strrep(curNames{i}{j},'.','_'))) % if
variable list not define OR if current signal found in specified variable list
            if (Interpolation)
                dataTable{last_index, 1} = curNames{i}{j};
            else
                dataTable{last_index, 1} = [curNames{i}{j} '_' timeVectors{i, 1}];
            end
            dataTable{last_index, 3} = timeVectors{i,2};
            try dataTable{last_index, 2} = read(mdfObj, i, mdfObj.ChannelNames{i}{j},
'OutputFormat','Vector');
            catch
                dataTableErrors(last_index, 1) = 1;
            end
            last_index = last_index + 1;
        end
    end
end
```

But the rest of this version of the code is overly simplified, particularly lacking error handling and feedback. Thus I combined the original `readMDF_DM_MD.m` and `Read_MDF_DM_AC.m` to update the code, making improvements and extensions:

- Supports fuzzy matching to filter based on the start, end, or specific patterns of signal names. This is achieved by adding a new `wildcard` function, which can handle variable lists marked with wildcards like `*`, as well as functions like `listStatsWith`, `listEndsWith`, and `listStartAndEndsWith`.
- Integrates a `waitbar`.
- Significantly improves error handling, e.g., capturing and displaying error messages when MDF file reading fails, then closing the waitbar and exiting the function.
- Adjusts details in handling overly long signal names, particularly managing character lengths more accurately when dealing with dot-separated names.
- Customizes shortening rules for signal names further through the use of regexp and contains functions, dynamically adjusting signal names based on specified name patterns in the listNames parameter.
- Extends data interpolation and time handling:
  - In interpolation mode, the code generates a new time vector to ensure all signals are interpolated at the same time points, which is crucial for subsequent data analysis.
  - Adds handling for POSIX time, including the initial timestamp's corresponding POSIX time in the final data table, which is beneficial for further analysis of time series data.
- Enhances code maintainability and extensibility:

- The function structure is clearer, and the introduction of error handling and user feedback mechanisms makes the code more robust and easier to maintain.
- The new fuzzy matching feature and progress bar mechanism allow for easy modifications or enhancements as needed.

Source code of new `Read_MDF_DM.m`:

```matlab
function [finalDatas,  FieldMatrix] = Read_MDF_DM(curPath, Interpolation, SampleTime,
SetStartTime0,listVars,listNames)
if nargin < 6 % listNames not defined
    listNames = {};
end
if nargin < 5 % listVars not defined
    listVars = {};
end

listStatsWith = cellfun(@(x) x(1:end-1),listVars(endsWith(listVars,'*')),'UniformOutput',false);
listEndsWith  = cellfun(@(x) x(2:end),listVars(startsWith(listVars,'*')),'UniformOutput',false);
listStartAndEndsWith = cellfun(@(x) x(2:end-
1),listVars(startsWith(listVars,'*')&endsWith(listVars,'*')),'UniformOutput',false);
listMatch     = listVars(~contains(listVars,'*'));


FieldMatrix = {};
finalDatas = {};
% Read_MDF

% [FileName,PathName,~] = uigetfile('*.mdf;*.mf4','MultiSelect','on');
warning('off','all');
curBarH = waitbar(0,'Parsing MDF...');
curBarHb=findobj(curBarH,'Type','figure');
curBarHt = get(get(curBarHb,'currentaxes'),'title');
try
    mdfObj = mdf(curPath);
catch ME
    warning(ME.message);
    fprintf('\t --> Ignored.\n');
    close(curBarH);
    return;
end

% Parse all available rasters
curNames = get(mdfObj, 'ChannelNames');
curSizes = cellfun(@length, curNames);
timeVectors = cell(length(curNames(:,1)), 2);
timeVectorsErrors = zeros(length(curNames(:,1)), 1);
dataTable = cell(sum(curSizes), 3);
dataTableErrors = zeros(sum(curSizes), 1);
last_index = 1;
waitbar(0.1,curBarH);

%% Modified for signal name more than 63 char
for i=1:length(curNames(:,1))
    timeVectors{i, 1} = ['t' num2str(i)];
    try timeVectors{i, 2} = read(mdfObj, i, mdfObj.ChannelNames{i}{curSizes(i)},
'OutputFormat','Vector');
        if isempty(timeVectors{i,2})
            timeVectorsErrors(i, 1) = 1;
        end
    catch
        timeVectorsErrors(i, 1) = 1;
    end

    for j=1:(curSizes(i) - 1)
        curSignalName = curNames{i}{j};
        if length(curSignalName) > 63 % Apply the method of 'Read_MDF_DM_AC.m'
```

```matlab
            if contains(curSignalName,'.')
                parts = strsplit(curSignalName,'.');
                parts = strrep(strrep(parts,']',''),'[','_');
                if (length(char(parts(end))) + length(char(parts(end-1))) < 62)
                    curSignalName = [char(parts(end-1)) '_' char(parts(end))];
                else
                    curSignalName = char(parts(end));
                end
            else
                curSignalName = curSignalName(end-62:end);
            end
        end

        curNames{i}{j} = strrep(curSignalName, '[', '_');
        curNames{i}{j} = strrep(curNames{i}{j}, ']', '');

        if isempty(listVars) ||  wildcard(strrep(curNames{i}
{j},'.','_'),listStatsWith,listEndsWith,listStartAndEndsWith,listMatch)
            if (Interpolation)
                dataTable{last_index, 1} = curNames{i}{j};
            else
                dataTable{last_index, 1} = [curNames{i}{j} '_' timeVectors{i, 1}];
            end
            dataTable{last_index, 3} = timeVectors{i,2};
            try dataTable{last_index, 2} = read(mdfObj, i, mdfObj.ChannelNames{i}{j},
'OutputFormat','Vector');
            catch
                dataTableErrors(last_index, 1) = 1;
            end
            last_index = last_index + 1;
        end
    end
    waitbar(0.1+i/length(curNames(:,1))*0.8,curBarH);
end

dataTable = dataTable(~dataTableErrors, :);
dataTable = dataTable(~cellfun(@isempty, dataTable(:, 1)), :);
timeVectors = timeVectors(~timeVectorsErrors, :);
timeVectors = timeVectors(~cellfun(@isempty, timeVectors(:, 1)), :);

curTimes2Print = cellfun(@(x,y)
[x,'_Length_of_t=',num2str(length(y))],timeVectors(:,1),timeVectors(:,2),'UniformOutput',false);

if (~Interpolation)
    [ChosenTimeIdx,~] = listdlg('ListString',curTimes2Print,'Name','Temps','ListSize',[300 400]);
    t=timeVectors{ChosenTimeIdx,2}(:,1);
    dataTable = dataTable(:,1:2);
    [~,idx] = sort(upper(dataTable(:,1)));
    finalDatas = [dataTable(idx, :) ; timeVectors ; {'t', t}];
else
    curBarHt.String = 'Interpolation...';
    ValidIdx = cellfun(@(x, y) ~isempty(x) && ~isempty(y) && length(x) == length(y), dataTable(:,2)
, dataTable(:,3));
    MaxT = max(cellfun(@(x) x(end), timeVectors(:,2)));
    MinT = min(cellfun(@(x) x(1), timeVectors(:,2)));
    NewTime = (MinT:SampleTime:MaxT)';
    dataTable = dataTable(ValidIdx, :);
    %     dataTable(:,3) = cellfun(@(x) x-x(1), dataTable(:,3), 'UniformOutput', false);
    idxUnitary = cellfun(@(x) length(x) == 1,dataTable(:,2));
    idxCell    = cellfun('isclass',dataTable(:,2),'cell');
    if (~isempty(find(idxUnitary,1)))
        unitTable = dataTable(idxUnitary & ~idxCell, :);
        unitTable(:,3) = cellfun(@(x) NewTime, unitTable(:,3), 'UniformOutput', false);
        unitTable(:,2) = cellfun(@(x) x*ones(length(NewTime), 1), unitTable(:,2), 'UniformOutput',
false);
    end

    dataTable = dataTable(~idxUnitary & ~idxCell, :);
```

```
%     dataTable(:,2) = cellfun(@(x, y) interp1(double(x), double(y), NewTime,
'linear',y(1)),dataTable(:,3), dataTable(:,2), 'UniformOutput', false);
    dataTable(:,2) = cellfun(@(x,y) interpCellFun(x,y,NewTime),dataTable(:,3), dataTable(:,2),
'UniformOutput', false);
    if (~isempty(find(idxUnitary & ~idxCell,1)))
        dataTable = [dataTable ; unitTable];
    end
    if (SetStartTime0)
        t = NewTime - NewTime(1);
    else
        t = NewTime;
    end
    dataTable = dataTable(:,1:2);
    [~,idx] = sort(upper(dataTable(:,1)));
    finalDatas = [dataTable(idx, :) ; {'t', t}];
end

%% adding for posixtime contained as initialTimestamp of mdfobj
% finalDatas = [finalDatas;{???????'t_posixtime',t+posixtime(mdfObj.InitialTimestamp)}???????];
if Interpolation
    finalDatas = [finalDatas;{'t_posixTime',posixtime(mdfObj.InitialTimestamp)+t}];
end

waitbar(1,curBarH);
close(curBarH)

warning('on','all');

% finalDatas(:,3) = cell(length(finalDatas(:,1)), 1);
if ~isempty(listNames)
    for ii=1:size(finalDatas,1)
        iFirstMatch = 0;
        nameFound = false;
        while ~nameFound && iFirstMatch<size(listNames,1)
            iFirstMatch = iFirstMatch+1;
            nameFound = contains(finalDatas{ii,1},listNames{iFirstMatch});
        end
        if nameFound
            finalDatas{ii,1} = finalDatas{ii,1}
(regexp(finalDatas{ii,1},listNames{iFirstMatch},'matchcase','once'):end);
        end
    end
end
finalDatas(:,1) = matlab.lang.makeValidName(finalDatas(:,1));
finalDatas(:,3) = num2cell(ones(length(finalDatas(:,1)) , 1));
end
% FUNCTIONS

% Wildcar function -> filter signals according to listvar
function outputBool = wildcard(sigList,listStatsWith,listEndsWith,listStartAndEndsWith,listMatch)
    startWithBool        = any(cellfun(@(x) any(startsWith(sigList,x)),listStatsWith));
    endWithBool          = any(cellfun(@(x) any(endsWith(sigList,x)),listEndsWith));
    startAndEndsWithBool = any(cellfun(@(x) any(startsWith(sigList,x))&
any(endsWith(sigList,x)),listStartAndEndsWith));
    listMatch            = any(cellfun(@(x) any(isequal(sigList,x)),listMatch));

    outputBool = startWithBool || endWithBool || startAndEndsWithBool || listMatch;
end
```

Extra-long signals are decoded with the full suffix preserved so that subsequent code can distinguish them:

```
>> who

Your variables are:

SUP_DBG_IFDBG_OUT_Output_data_Dev_FUSION3_EgolaneLeftlineLength
SUP_DBG_IFDBG_OUT_Output_data_Dev_FUSION3_EgolaneLeftline_c0
SUP_DBG_IFDBG_OUT_Output_data_Dev_FUSION3_EgolaneLeftline_c1
SUP_DBG_IFDBG_OUT_Output_data_Dev_FUSION3_EgolaneLeftline_psi0
```

```
SUP_DBG_IFDBG_OUT_Output_data_Dev_FUSION3_EgolaneLeftline_x0
SUP_DBG_IFDBG_OUT_Output_data_Dev_FUSION3_EgolaneLeftline_y0
SUP_DBG_IFDBG_OUT_Output_data_Dev_FUSION3_EgolaneRightline_c0
SUP_DBG_IFDBG_OUT_Output_data_Dev_FUSION3_EgolaneRightline_c1
SUP_DBG_IFDBG_OUT_Output_data_Dev_FUSION3_EgolaneRightline_psi0
SUP_DBG_IFDBG_OUT_Output_data_Dev_FUSION3_EgolaneRightline_x0
SUP_DBG_IFDBG_OUT_Output_data_Dev_FUSION3_EgolaneRightline_y0
SUP_DBG_IFDBG_OUT_Output_data_Dev_FUSION3_previousT0obj_ax
SUP_DBG_IFDBG_OUT_Output_data_Dev_FUSION3_previousT0obj_ay
SUP_DBG_IFDBG_OUT_Output_data_Dev_FUSION3_previousT0obj_class
SUP_DBG_IFDBG_OUT_Output_data_Dev_FUSION3_previousT0obj_vx
SUP_DBG_IFDBG_OUT_Output_data_Dev_FUSION3_previousT0obj_vy
SUP_DBG_IFDBG_OUT_Output_data_Dev_FUSION3_previousT0obj_x
SUP_DBG_IFDBG_OUT_Output_data_Dev_FUSION3_previousT0obj_y
SUP_DBG_IFDBG_OUT_Output_data_Dev_FUSION3_previousT0obj_yaw
......
```